

Frankfurt am Main
Johann Wolfgang Goethe-Universität

Fachbereich 12 - Informatik

Verbessern der Pulsarerkennung durch Optimierung der Filter-Initialisierung von CNNs

BACHELORARBEIT
IM STUDIENGANG INFORMATIK (B. SC.)

VORGELEGT VON

Jad Dayoub

MATRIKELNUMMER: 7425569

BETREUER:

Prof. Dr. Volker Lindenstruth

ZWEITGUTACHTER:

Prof. Dr. Arne Nägel

ABGABEDATUM: 27. AUGUST 2024

ZUSAMMENFASSUNG

Pulsare sind rotierende Neutronensterne, die in regelmäßigen Abständen Lichtimpulse abgeben. Diese können auf der Erde anhand von Radioteleskopen gemessen werden. Aufgrund des hohen Datenaufkommens während der Online-Datenverarbeitung ist es nicht möglich, die Daten in Echtzeit ausreichend zu bewerten. Da die Entscheidungen für das Speichern der Daten anhand unvollständiger Informationen getroffen werden, führt das zu einem irreversiblen Datenverlust. Um das zu minimieren, sollten die Daten während der Datenerfassung identifiziert werden, um fundierte Entscheidungen über ihre Speicherung treffen zu können. Eine mögliche Herangehensweise ist durch die Nutzung von gefalteten neuronalen Netzwerken (englisch: „Convolutional Neural Network“, kurz CNN).

Im Rahmen dieser Arbeit wird die Erkennung von Pulsaren anhand von synthetisierten Daten der Machine Learning-basierten Pipeline für die Analyse von Pulsaren (ML-PPA) verbessert, die vom PUNCH4NFDI-Konsortium entwickelt wurde. Das ML-PPA Software-Framework stellt sich der Herausforderung, Pulsarsignale aus großvolumigen Datenströmen mithilfe von Methoden des Maschinellen Lernens zu identifizieren.

Ziel dieser Arbeit ist es, die Erkennung von Pulsaren anhand von CNNs zu verbessern. Dabei wird das Problem der zufälligen Filter-Initialisierung untersucht, da sie die Leistungsfähigkeit des Netzwerks erheblich beeinträchtigen kann. Um diese Herausforderung zu adressieren, werden vordefinierte Kantenerkennungsfilter wie die Sobel-, Prewitt- und Kirsch-Filter in den CNN implementiert, um deren Auswirkungen auf die Konvergenz und Genauigkeit des Netzwerks eingehend zu analysieren. Durch eine systematische Evaluierung dieser Filter mit verschiedenen Bildgrößen zeigt sich, dass die Anwendung von Kirsch-Filttern für kleinere Bilder und Prewitt sowie Sobel für größere Bilder zu einer stabilen und effizienten Erkennung von Pulsaren führt. Die Filter ermöglichen es dem CNN, schneller zu konvergieren und relevante Merkmale in den Daten effizient zu extrahieren. Die Ergebnisse der Arbeit zeigen, dass vordefinierte Kantenerkennungsfilter die Konvergenzgeschwindigkeit bei der Erkennung von Pulsaren erhöhen und eine konsistente Merkmalsextraktion bieten, jedoch bei komplexeren Datensätzen mit zusätzlichen Interferenzsignalen keine signifikanten Verbesserungen der Genauigkeit bewirken. Um diese Herausforderung zu bewältigen, wurde eine vereinfachte Variante des Canny-Algorithmus eingesetzt, der sich als effektiv erwies und bessere Ergebnisse bei komplexeren Datensätzen mit zusätzlichen Interferenzsignalen lieferte.

Inhaltsverzeichnis

1 Motivation	1
2 Grundlagen	3
2.1 Pulsare	3
2.2 Künstliche neuronale Netzwerke	4
2.3 Gefaltete neuronale Netzwerke	6
2.4 Kantenidentifikation in der Bildbearbeitung	9
3 Methodik	11
3.1 Datenerhebung	11
3.2 Angewendete Filter	13
3.2.1 Standard-Filter	13
3.2.2 Prewitt-Filter	14
3.2.3 Sobel-Filter	15
3.2.4 Kirsch Kompass Maske	16
3.3 Netzwerk Architektur und Test Aufbau	18
4 Ergebnis	21
4.1 Erkennung von Pulsaren	21
4.2 Erweiterter Datensatz	24
4.3 Anwendung des Canny-Algorithmus	27
5 Fazit	29
6 Literatur	31
Abbildungsverzeichnis	33

1 Motivation

Im Bereich der Astrophysik spielen Pulsare eine entscheidende Rolle bei der Erforschung des Universums. Bei diesen kosmischen Objekten handelt es sich um rotierende Neutronensterne, die in regelmäßigen Abständen Lichtimpulse aussenden. Sie wurden 1967 von Jocelyn Bell und Antony Hewitt bei der Untersuchung des Himmels nach Radiosignalen entdeckt, da ihre Strahlung für gewöhnlich im Radiofrequenzbereich liegt [1]. Die Identifizierung der Pulsare ist von großer Bedeutung bei der Erforschung von Gravitationswellen sowie zur Untersuchung der allgemeinen und speziellen Relativitätstheorie [2]. Allerdings stellt ihre Erkennung eine Herausforderung dar, da es bei der Datenerfassung durch Radioteleskope nicht möglich ist, den eingehenden Datenstrom vollständig in Echtzeit zu bewerten. Aufgrund der begrenzten Zeit, die für die Analyse der eingehenden Daten zur Verfügung steht, erfolgt die Bewertung der Daten anhand unvollständiger Informationen. Dies führt dazu, dass Entscheidungen über die Speicherung getroffen werden, bevor eine umfassende Analyse durchgeführt werden kann und zu einem irreversiblen Datenverlust führt. Zusätzlich können beim Erfassen der Daten Störsignale durch verschiedene Quellen auftreten [3].

Als eine mögliche Strategie zur Lösung dieses Problems werden in der vorliegenden Arbeit künstliche neuronale Netzwerke verwendet, die mit synthetisierten Daten der Machine Learning-basierten Pipeline für die Analyse von Pulsaren (ML-PPA) trainiert werden [3]. ML-PPA ist ein Software-Framework, das sich der Herausforderung stellt, Pulsarsignale aus großvolumigen Datenströmen während der Datenerfassungsphase anhand Methoden des Maschinellen Lernens zu identifizieren. Das Framework soll Astronomen in ihrem Streben nach der Entdeckung astronomischer Signale unterstützen und ihre Möglichkeiten zur Analyse und Interpretation umfangreicher astronomischer Daten verbessern. Es wurde vom PUNCH4NFDI-Konsortium entwickelt, das sich aus Experten diverser, deutschlandweiter Universitäten und Instituten, wie der Max Planck Gesellschaft, der Deutschen Physikalische Gesellschaft (DPG), dem Europäischen Kernforschungszentrum (CERN) und der Goethe-Universität Frankfurt zusammensetzt. Das Konsortium arbeitet im Rahmen der Nationalen Forschungsdateninfrastruktur (NFDI) an der Entwicklung und Implementierung fortschrittlicher Methoden zur Verwaltung großer wissenschaftlicher Datenmengen. Das Akronym „PUNCH“ steht hierbei für „Particles, Universe, NuClei, and Hadrons“ (auf deutsch: Teilchen, Universum, Nukleii und Hadronen) [4].

Besonders leistungsfähig für die Verarbeitung von Bilddaten sind gefaltete neuronale Netzwerke (englisch „Convolutional Neural Network“, CNN), eine spezielle Form neuronaler Netzwerke, die in zahlreichen Anwendungsbereichen erfolgreich eingesetzt werden. Insbesondere in der Bildklassifizierung ermöglichen sie eine zuverlässige Erkennung, da sie Strukturen unabhängig von ihrer räumlichen Anordnung identifizieren können [5]. Ein entscheidender Faktor für den Erfolg von neuronalen Netzwerken ist die Gewichtsinitialisierung, da sie direkten Einfluss auf die Konvergenz des Netzwerkes hat und somit die Leistungsfähigkeit des Netzwerks maßgeblich beeinflusst [6].

Das zentrale Ziel dieser Arbeit besteht darin, zu untersuchen, ob die Verwendung vordefinierter Filter in CNNs die Erkennung von Pulsaren gegenüber standardmäßig zufällig initialisierten Filtern verbessern kann. Durch eine systematische Evaluierung des Einsatzes vordefinierter Filter wird untersucht, ob und

inwiefern diese Anpassung zu einer präziseren und effizienteren Erkennung von Pulsaren führt. In diesem Zusammenhang werden die Prewitt-, Sobel- und Kirsch-Filter implementiert und getestet.

Im Verlauf der Arbeit werden in Kapitel 2 die notwendigen Grundlagen zur Datenerhebung von Pulsaren sowie die Funktionsweise von CNNs erläutert. Dabei werden ebenfalls die theoretischen Grundlagen hinter den Filtern behandelt.

Kapitel 3 stellt die verwendeten Daten und Filter vor, die für das Testen genutzt werden. Zusätzlich wird die Architektur des eingesetzten CNNs und der Test Aufbau dargelegt.

Im vierten Kapitel werden die Ergebnisse bei Anwendung der Filter analysiert, wobei zunächst die Unterscheidung zwischen Pulsaren und Nicht-Pulsaren vorgenommen wird. Anschließend erfolgt die erweiterte Erkennung von Radiointerferenzen.

Kapitel 5 enthält das Fazit der Arbeit.

2 Grundlagen

Dieses Kapitel behandelt die notwendigen Grundlagen zum Verständnis der Problemstellung. Zunächst werden die theoretischen Grundlagen zu Pulsaren sowie ihrer Datenerhebung erläutert. Anschließend werden neuronale Netzwerke und Convolutional Neural Networks erklärt, um auf die Funktionsweise von Filtern einzugehen. Abschließend wird die Bedeutung der Filter in der Bildbearbeitung sowie ihre Rolle in CNNs behandelt.

2.1. Pulsare

Wenn ein Stern am Ende seines Lebens in einer Supernova explodiert und anschließend in sich zusammenstürzt, bildet sich bei einer Gesamtmasse vom 1,44- bis 3-fache unserer Sonnenmasse daraus ein Neutronenstern [7]. Aufgrund der Drehimpulserhaltung behält dieser seine anfängliche Rotationsgeschwindigkeit bei und dreht sich daher mit einer hohen Geschwindigkeit, wodurch ein Magnetfeld entsteht. Liegen die Rotationsachse und die Achse des Magnetfeldes nicht aufeinander, rotiert ebenfalls das Magnetfeld, wodurch an den Polen gebündelte Strahlung abgegeben wird. Trifft dieser Strahl die Erde, ergibt sich ein wiederkehrendes Signal, das für gewöhnlich im Radiofrequenzbereich liegt. Ein solch rotierender Neutronenstern wird als „Pulsar“ bezeichnet [1].

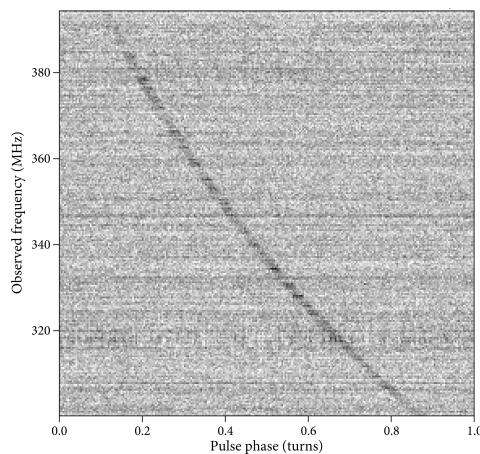


Abbildung 2.1: Beispiel für die Ausbreitungsverzögerung eines Pulses von J1800+0534 [3].

Eins der wesentlichen beobachtbaren Phänomene, das besonders bei der Identifizierung von Pulsaren eine Rolle spielt, ist die Dispersionsverzögerung [3]. Dabei handelt es sich um eine Verzögerung der frequenzabhängigen Geschwindigkeit des Pulsarsignals, verursacht durch die Interaktion des Signales mit dem interstellaren Medium (ISM). Dadurch propagieren höherfrequente Radiowellen schneller durch das Medium als niederfrequente. Dieses Phänomen ist in Abbildung 2.1 dargestellt. Die charakteristische Frequenzabhängigkeit der Ankunftszeiten zeigt sich als schräge Linie im Diagramm, wobei die vertikale Achse die

beobachtete Frequenz und die horizontale Achse die Phase des Pulsarsignals in Umdrehungen darstellt. Die Phase beschreibt den Fortschritt des Signals innerhalb einer Periode des Pulsars als normierte, proportionale Darstellung der Zeit.

$$\Delta t = 4.12\text{ms} ((f_{\text{LO}}[\text{GHz}])^{-2} - (f_{\text{HI}}[\text{GHz}])^{-2}) \text{ DM}[\text{cm}^{-3}\text{pc}] \quad (2.1)$$

Gleichung 2.1 gibt die Zeitverzögerung zweier Frequenzkanäle, f_{LO} und f_{HI} , an, aus der die Frequenz-Zeit-Messung resultiert. Das Dispersionsmaß (DM) ist hierbei ein wichtiger Parameter, der die Elektronendichte im ISM entlang der Sichtlinie zwischen Pulsar und Erde quantifiziert. Sie deutet auf die Stärke der Verzögerung des Signals hin und ist definiert als das Produkt der durchschnittlichen Elektronendichte n_e pro Kubikmeter und der Distanz D zwischen Pulsar und Erde in Parsecs:

$$\text{DM} = n_e[\text{cm}^{-3}] \text{ D}[pc] \quad (2.2)$$

2.2. Künstliche neuronale Netzwerke

In dieser Arbeit werden gefaltete neuronale Netzwerke für die Verarbeitung der Pulsardaten verwendet. Dafür werden in diesem Abschnitt die grundlegenden Konzepte neuronaler Netze erklärt.

Künstliche neuronale Netze sind ein Teilbereich des Maschinellen Lernens und bilden das Herzstück von „Deep-Learning“-Algorithmen. Sie wurden dem menschlichen Gehirn nachempfunden, da sie die Struktur und Funktionsweise biologischer Neuronen nachahmen. Sie sind in der Lage, durch die Anpassung ihrer internen Parameter, komplexe Muster in umfangreichen Datenmengen zu erkennen und zu modellieren. Ihre Architektur besteht aus einer großen Anzahl von Knoten, die auch als Neuronen bezeichnet werden. Diese Neuronen sind in Schichten angeordnet, wobei jedes Neuron einer Schicht mit allen Neuronen der vorherigen und der darauffolgenden Schicht verbunden ist. Diese Schichten werden auch als „vollständig verbundene Schichten“ bezeichnet. Jede der Verbindungen besitzt ein Gewicht, das bestimmt, wie stark die Ausgabe eines Neurons das nächste Neuron beeinflusst. Bei den Gewichten handelt es sich um anpassbare Parameter des Netzwerks, die im Verlauf des Trainingsprozesses optimiert werden, um das Netzwerk auf die Daten auszurichten. Auf die Ausgabe eines Neurons wird eine Aktivierungsfunktion angewendet, die bestimmt, ob und in welchem Maße das Neuron seine Information an die nächste Schicht weitergibt. Diese Funktion ist entscheidend für die Fähigkeit des Netzes, nicht-lineare Zusammenhänge zu erkennen. Die Ausgabe nach Anwendung einer Aktivierungsfunktion bezeichnet man als „Aktivierung des Neurons“ [5].

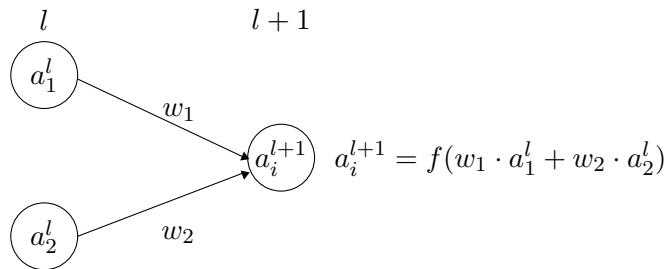


Abbildung 2.2: Aktivierung a_i eines Knotens i in Schicht $l + 1$ mit einer Aktivierungsfunktion f .

Abbildung 2.2 zeigt den schematischen Aufbau eines kleinen Abschnitts eines neuronalen Netzwerks. Die Abbildung stellt die Aktivierung eines Knotens a_i^{l+1} einer Schicht $l + 1$ dar, welcher die gewichteten Aktivierungen der Knoten a_1^l und a_2^l der vorherigen Schicht l empfängt und auf die Summe eine Aktivierungsfunktion f anwendet.

Eine typische Aktivierungsfunktion ist die Rectified Linear Unit (ReLU), die alle negativen Eingaben auf 0 setzt, während sie positive Werte unverändert lässt. Dadurch kann das Netzwerk komplexe, nicht-lineare Abhängigkeiten erkennen.

Es werden grundsätzlich zwischen drei Arten von Schichten unterschieden: der Eingabeschicht, der Ausgabeschicht und den verborgenen Schichten [5].

Die Eingabeschicht (englisch: „Input Layer“) steht am Anfang des Netzes und nimmt die Rohdaten als numerische Werte entgegen, die für jeden Eingabewert ein Neuron hat.

Zwischen der Eingabe- und Ausgabeschicht befinden sich die verborgenen Schichten (englisch: „Hidden Layer“) und führen dort die eigentlichen Berechnungen durch. Sie empfangen die Ausgaben der vorherigen Schicht, verarbeiten diese und leiten das Ergebnis an die nächste Schicht weiter. Jedes Neuron in einer verborgenen Schicht berechnet eine gewichtete Summe der Aktivierungen der vorherigen Schicht und addiert gegebenenfalls zusätzlich einen Bias-Wert hinzu. Sie ermöglicht es dem Netzwerk, die Aktivierungen der Neuronen zu verschieben, wodurch es in der Lage ist, Muster zu erfassen, die nicht allein durch die Eingangswerte erklärt werden können.

Die Ausgabeschicht (englisch: „Output Layer“) befindet sich am Ende und gibt die endgültige Vorhersage aus. Sie verarbeitet die Informationen aus den verborgenen Schichten und stellt sie in einer Form bereit, die für die spezifische Aufgabe geeignet ist.

Mit jeder zusätzlichen Schicht wird das Netzwerk tiefer, wodurch es ihm ermöglicht wird, komplexere Merkmale aus den Daten zu extrahieren und zu verarbeiten, und ihn somit als Deep-Learning-Algorithmus charakterisiert. Ein beispielhafter Aufbau eines neuronalen Netzes, der über zwei verborgenen Schichten Datenpunkte auf zwei Klassen klassifiziert, ist in Abbildung 2.3 zu sehen.

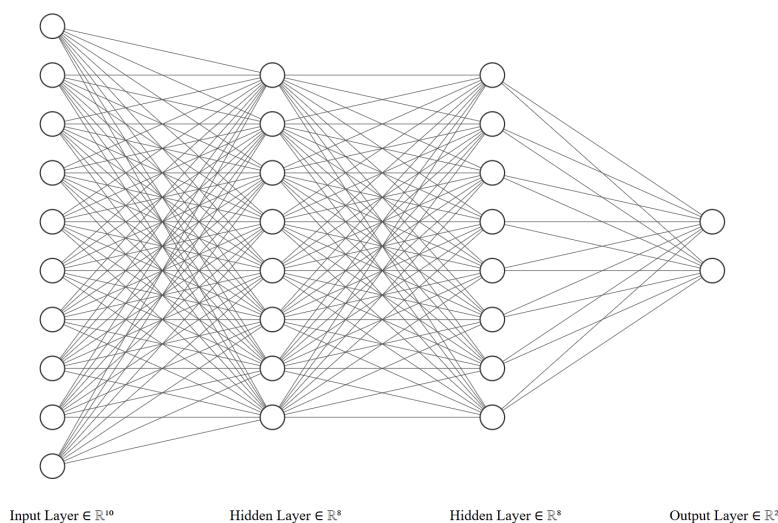


Abbildung 2.3: Beispiel für ein neuronales Netz mit 2 verborgenen Schichten.

Durch das Trainieren eines neuronalen Netzwerks auf einen Datensatz lernt es, grundlegende Muster und Zusammenhänge in den Daten zu erkennen. Das Trainieren eines neuronalen Netzes zielt dabei auf die Optimierung der Gewichte und Bias-Werte der eingehenden Daten ab, sodass die Vorhersagen des Netzes möglichst genau mit den tatsächlichen Ergebnissen übereinstimmen. Dabei wird anhand einer Verlustfunktion die Abweichung zwischen den beiden Werten quantifiziert, um den Fehler zu bestimmen. Eine gängige Verlustfunktion für Klassifikationsprobleme ist die Kreuzentropie. Bei dieser Funktion ist der Verlust geringer, je näher die Vorhersage an der tatsächlichen Klasse liegt und höher, wenn die Vorhersage weit von der eigentlichen Klasse abweicht [5].

Um das Netzwerk zu verbessern, muss der Fehler über alle Datenpunkte eines Datensatzes minimiert werden. Das geschieht durch einen Optimierungsalgorithmus, der den Gradienten der Verlustfunktion in Bezug auf jedes Gewicht im Netzwerk berechnet, um die Gewichte an die gegebenen Daten anzupassen. Der grundlegende Ansatz dafür ist der Gradientenabstieg (englisch: „Gradient Descent“). Bei diesem Verfahren werden die Gewichte schrittweise angepasst, indem sie proportional zur negativen Richtung des Gradienten der Verlustfunktion aktualisiert werden. Der Anpassungsschritt wird durch die Lernrate bestimmt, die die Größe der Schritte angibt. Eine angemessene Wahl der Lernrate ist entscheidend: ist die Lernrate zu hoch, kann das Netzwerk die optimale Lösung überspringen; ist sie zu niedrig, verlangsamt sich die Konvergenz erheblich und das Netzwerk könnte in einem lokalen Minimum stecken bleiben. Diese beiden Probleme werden als „Explodieren“ und „Verschwinden des Gradienten“ bezeichnet [8]. Konvergenz bezeichnet hier den Prozess, bei dem die Gewichte des Netzwerks während dem Training schrittweise optimiert werden, sodass sich die Verlustfunktion stabilisiert und die Vorhersagen des Netzwerks zunehmend genauer werden. Eine häufig verwendete Erweiterung von Gradient Descent ist der Adam-Optimizer (Adaptive Moment Estimation), der adaptive Lernraten für jedes Gewicht verwendet [9]. Er berücksichtigt dabei nicht nur den aktuellen Gradienten, sondern auch die ersten beiden Momente, den Mittelwert und die Varianz, der Gradienten. Dadurch kann der Adam-Optimizer automatisch die Schrittweite anpassen, wodurch eine schnellere und stabilere Konvergenz erreicht wird.

Backpropagation ist der zentrale Algorithmus zur effizienten Berechnung der Gradienten. Dabei wird der Fehler von der Ausgabeschicht des Netzes bis hin zur Eingabeschicht zurückpropagiert. Für jede Schicht wird der Gradient der Verlustfunktion in Bezug auf die Gewichte dieser Schicht berechnet. Dies geschieht durch Anwendung der Kettenregel der Differenzialrechnung, die es ermöglicht, die Ableitung der Verlustfunktion hinsichtlich jedes Gewichts als Produkt einfacherer Ableitungen zu zerlegen [5].

Das Training erfolgt in mehreren Epochen, in denen das Netzwerk den gesamten Trainingsdatensatz durchläuft. Um das Training effizienter zu gestalten, wird der Datensatz in kleinere Stapel (englisch: „Batches“) unterteilt. Ein Batch ist eine Teilmenge der Trainingsdaten, die in einem Schritt verarbeitet wird. Nach jedem Batch wird der Fehler berechnet und die Gewichte werden sofort angepasst, wodurch der Lernprozess stabiler und schneller wird. Über viele Epochen hinweg verfeinert das Netzwerk so seine Vorhersagen, indem es kontinuierlich die Fehler minimiert. Die Leistung des Netzwerks wird anschließend anhand von ungesehenen Testdaten bewertet.

2.3. Gefaltete neuronale Netzwerke

In vollständig verbundenen Netzwerken werden Bilder als eindimensionale Vektoren eingegeben. Das führt zu Schwierigkeiten bei der Verarbeitung, da die räumliche Struktur der Bildmerkmale nicht berücksichtigt wird. Um die Pulsardaten optimal verarbeiten zu können, werden gefaltete neuronale Netzwerke (englisch:

„Convolutional Neural Network“, CNN) verwendet, eine spezielle Art neuronaler Netzwerke, die dieses Problem löst. Sie wurden erstmals 1998 von LeCun vorgestellt und sind besonders effektiv bei der Verarbeitung und Analyse von Bilddaten [10].

CNNs sind darauf ausgelegt, gitterartige Datenstrukturen zu verarbeiten. Ihre Hauptanwendung finden sie in der Bildverarbeitung, da Bilder als zweidimensionales Gitter aus Pixeln betrachtet werden können [11]. Sie bestehen aus zwei wesentlichen Schichttypen: Faltungsschichten und Pooling-Schichten.

Der zentrale Baustein eines CNNs ist die Faltungsschicht. Sie extrahiert wichtige Merkmale aus den Eingabedaten, indem sie eine Faltung des Bildes mit einem sogenannten Filter durchführt. Ein Filter ist eine kleine, quadratische Matrix von Gewichten, die über das gesamte Bild bewegt wird, um eine Merkmalskarte (englisch: „Feature Map“) zu erzeugen. Jede Position des Filters erzeugt einen Wert in der Feature Map, der die Aktivierung eines spezifischen Merkmals an der entsprechenden Position im Eingabebild widerspiegelt. Die Gewichte werden im Verlauf des Trainingsprozesses an die gegebenen Daten angepasst, um die Merkmale effizient zu extrahieren. Die mathematische Operation, die in dieser Schicht durchgeführt wird, ist die Faltung. Sie führt beim Bewegen des Filters über das Bild eine elementweise Multiplikation und anschließende Summation der resultierenden Werte durch. Jeder Filter einer Faltungsschicht erzeugt eine Feature Map, die der nächsten Faltungsschicht als Eingabe übergeben werden. Diese Faltung ermöglicht es dem CNN, lokale Merkmale wie Kanten, Texturen und andere Muster zu erkennen. In Abbildung 2.4 wird der Prozess der Faltung anschaulich dargestellt.

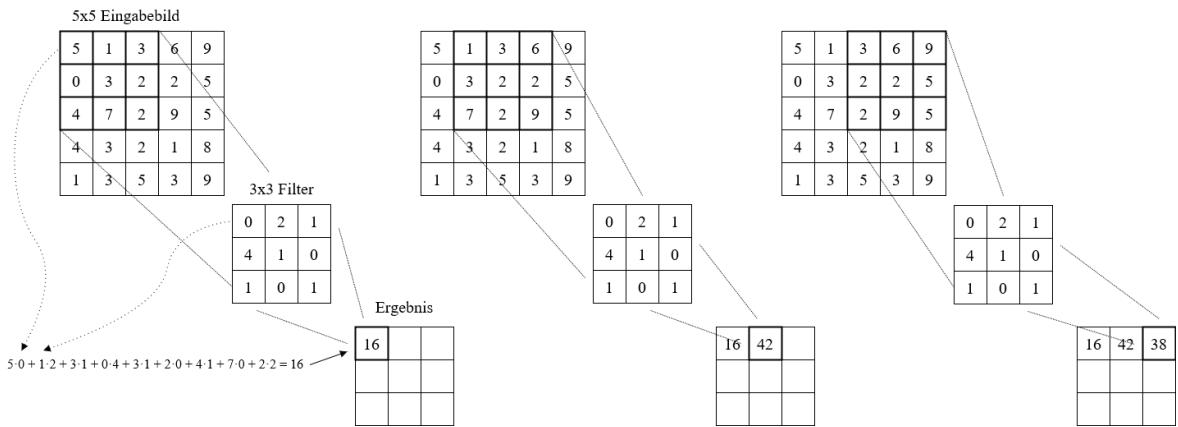


Abbildung 2.4: Beispiel für die Faltung eines Bildes der Größe 5x5 mit einem Filter der Größe 3x3.

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n) \quad (2.3)$$

Die Faltung eines Bildes I mit einem Filter K ist in Gleichung 2.3 definiert, bei der $S(i, j)$ für einen Pixel im Ergebnisbild S steht [11]. Jeder Pixel (i, j) im Ergebnisbild S ist das Resultat der gewichteten Summe der benachbarten Pixel im ursprünglichen Bild I , wobei die Gewichte durch den Filter K bestimmt werden.

Da die Faltung kommutativ ist, lässt sich die Reihenfolge der Operanden vertauschen:

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n) \quad (2.4)$$

Dabei wird der Filter spiegelverkehrt und entlang beider Achsen rotiert angewendet. In den meisten Bibliotheken für neuronale Netzwerke wird eine ähnliche Funktion namens „Kreuzkorrelation“ implementiert (siehe Gleichung 2.5). Sie entspricht der Faltungsoperation ohne das Rotieren des Filters [11], wird allerdings dennoch als Faltung betitelt und im Verlaufe der Arbeit ebenfalls unter dem Begriff geführt.

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n) \quad (2.5)$$

Die Anwendung der Kreuzkorrelation ist in Abbildung 2.4 anschaulich dargestellt. Dabei wird ein 3x3 Filter über ein Bild der Größe 5x5 bewegt, um in jedem Schritt eine elementweise Multiplikation und anschließende Summation durchzuführen.

Bei einer Bildgröße von (x, y) und einer Filtergröße von (m, n) ergibt sich für das Ergebnisbild eine Größe von (x - m + 1, y - n + 1). Dadurch gehen allerdings einige Pixel verloren, da der Filter am Rand des Bildes nicht angewendet werden kann und die Faltung an dieser Stelle verworfen wird. Bei einer Filtergröße von 3x3 würden dadurch jeweils 2 Zeilen und Spalten an Pixeln verloren gehen. Für große Bilder stellt das keinen relevanten Verlust dar. Zum Beispiel hätte ein 4K UHD Bild mit einer Auflösung von 3840x2160 nach der Faltung mit einem 3x3 Filter eine Größe von 3838x2158, das einem Pixelverlust von ungefähr 0.1% entspricht.

Bei kleineren Bildern, größeren Filtern oder mehrfacher Faltung aufgrund tieferer CNNs, wird der Verlust allerdings größer. Das kann mit Padding umgangen werden, indem vor der Faltung Pixel hinzugefügt werden, wodurch das Falten an den Rändern möglich ist. Dabei werden die Ränder mit entsprechend vielen Zeilen und Spalten erweitert, für gewöhnlich mit einem Pixelwert von 0, um die Größe des Bildes nach der Faltung mit einem Filter nicht zu verändern [13].

Nach der Faltungsschicht folgt in der Regel eine Pooling-Schicht, die zur Dimensionsreduktion der Feature Map dient. Das Pooling vereinfacht die Feature Map, indem es die Informationen auf das Wesentliche reduziert und gleichzeitig die Erkennung wesentlicher Merkmale unabhängig von ihrer genauen Position ermöglicht. Dabei wird eine Pooling-Operation auf lokale Bereiche der Feature Map angewendet, indem ein Pooling-Fenster über das Bild bewegt wird. Ein typisches Pooling-Fenster hat dabei eine Größe von 2x2. Die Schrittweite wird hier für gewöhnlich so gewählt, dass sich die Bereiche, auf die das Fenster angewendet wird, nicht überlappen.

Eine gängige Methode ist das Max-Pooling, bei dem der höchste Wert innerhalb des Pooling-Fensters ausgewählt wird. Abbildung 2.5 veranschaulicht diesen Prozess, indem die verschiedenen Positionen des Pooling-Fensters farblich dargestellt sind. Dadurch wird die Komplexität des Netzwerks reduziert, ohne die wesentlichen Merkmale zu verlieren. Nach jeder Faltungs- oder Pooling-Schicht wird eine Aktivierungsfunktion wie ReLU angewendet, um nichtlineare Zusammenhänge zu modellieren.

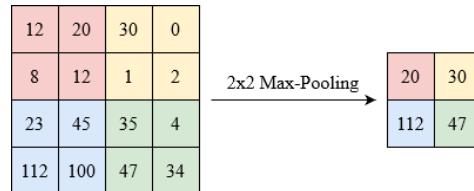


Abbildung 2.5: Beispiel für 2x2 Max-Pooling [12].

Am Ende eines CNNs stehen eine oder mehrere vollständig verbundene Schichten, die die extrahierten Merkmale verwenden, um daraus eine Klassifizierung vorzunehmen. Die letzte Schicht eines derartigen Netzwerks nutzt in der Regel die Softmax-Aktivierungsfunktion, die eine Wahrscheinlichkeitsverteilung über die möglichen Klassen der Eingabedaten erzeugt.

2.4. Kantenidentifikation in der Bildbearbeitung

Das ursprüngliche Anwendungsgebiet von Filtern liegt in der Bildbearbeitung, insbesondere zur Verbesserung, Analyse und Transformation von Bildern genutzt werden. In diesem Kontext lassen sich Filteroperationen in zwei Hauptkategorien unterteilen: in Operationen, die Informationen aus bestehenden Bildern extrahieren, und in Operationen, die neue Bilder erzeugen. Besonders relevant für diese Arbeit sind die Filteroperationen, die zur Erzeugung neuer Bilder führen.

Bei der Erzeugung eines neuen Bildes werden grundsätzlich drei Arten von Operationen unterschieden: Punktoperationen, globale Operationen und Nachbarschaftsoperationen [14]. Punktoperationen berechnen einen neuen Pixelwert allein in Abhängigkeit von seinem bisherigen Pixelwert, bspw. bei Helligkeitsanpassungen. Globale Operationen hingegen verändern das Bild in ihrer grundlegenden Struktur, bspw. bei Skalierung oder Rotation.

Nachbarschaftsoperationen betrachten sowohl den Pixelwert des aktuellen Pixels als auch die Werte seiner unmittelbaren Nachbarn, und berechnen daraus einen neuen Pixelwert an der selben Koordinate wie der Referenzpunkt. Eine gängige Form von Nachbarschaftsoperationen sind Faltungsoperationen. Dabei wird ein Filter auf das Bild angewendet, um lokale Bildmerkmale zu berechnen. Faltungsoperationen können genutzt werden, um ein Bild zu glätten, Rauschen zu entfernen oder Kanten hervorzuheben.

Bei Kanten in einem Bild handelt es sich um eine abrupte Änderung der Intensität der Pixelwerte [14]. Abbildung 2.6 zeigt eine beispielhafte Darstellung einer Kante. Derartige Änderungen entstehen typischerweise dort, an denen sich die Eigenschaften eines Objekts im Bild stark vom Hintergrund abheben. In der Astronomie, und speziell bei der Analyse von Signalen von Pulsaren, ist dies von besonderer Bedeutung. Pulsare erzeugen im Bild charakteristische Signaturen, die oft als helle Kurven oder Linien dargestellt werden. Diese Signaturen heben sich durch eine deutlich höhere Intensität von ihrem Umfeld ab, die in der Bildverarbeitung als Kante interpretiert wird.

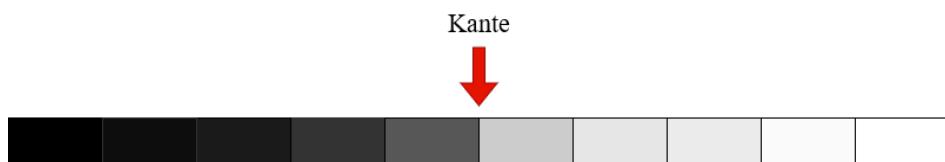


Abbildung 2.6: Beispiel einer Kante als Pixelintensitätsänderung im Bild. Dabei hat die geringste Intensität (schwarz) den Wert -1 und die höchste (weiß) den Wert 1 [15].

Um Kanten zu identifizieren, wird oft die erste Ableitung des Pixelintensitätsgraphen genommen. Diese Ableitung zeigt die Steigung der Intensitätskurve an jedem Punkt an, die in einem Bild als Veränderung oder Kante sichtbar wird. In Abbildung 2.7 ist der Pixelintensitätsgraph der Kante aus Abbildung 2.6 als blaue Kurve dargestellt. Die abrupte Änderung der Intensität ist in dem Graphen als steilen Anstieg sichtbar. Die in orange dargestellte Ableitungskurve zeigt die Position der größten Intensitätsänderung an.

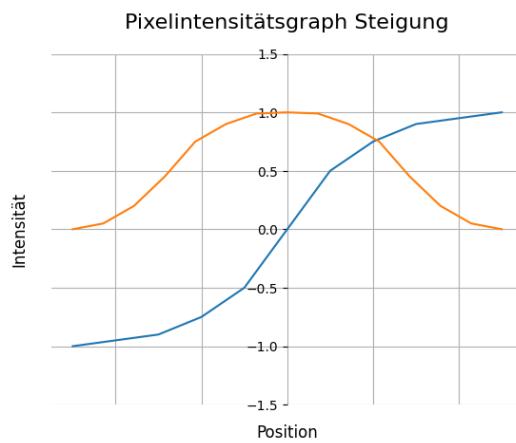


Abbildung 2.7: Pixelintensitätsgraph (blau) der Kante aus Abbildung 2.6 mit Steigung der Kurve an jedem Punkt (orange), inspiriert von [15].

Im Gegensatz zur klassischen Bildbearbeitung, bei der Filter manuell definiert und angewendet werden, lernt ein CNN die optimalen Filter im Trainingsprozess selbstständig

Die ersten Schichten eines CNNs dienen in der Regel der Erkennung grundlegender Merkmale wie Kanten, Ecken und einfachen Texturen. Sie berechnen, ähnlich wie traditionelle Filter, die erste Ableitung der Pixelintensitäten, um Kanten hervorzuheben.

3 Methodik

In diesem Kapitel wird die Herangehensweise zur Lösung des Problems dargelegt. Beginnend mit einer Erklärung zur Erstellung der synthetisierten Daten, gefolgt von der Vorstellung der genutzten Filter und der Netzwerkarchitektur, sowie dem Aufbau der Test.

3.1. Datenerhebung

Während der Messung nach Pulsaren entsteht häufig eine erhebliche Dysbalance zwischen den verschiedenen Signalarten auf. Um dieses Ungleichgewicht zu kompensieren und die Trainingsbedingungen zu verbessern, werden synthetisierte Daten verwendet. Sie ermöglichen die Erstellung eines ausgewogenen Datensatzes, der dem Netzwerk hilft, präziser zu trainieren und die Signale effektiver zu erkennen.

Die Datenerhebung erfolgt durch die Verwendung der „Machine Learning-basierten Pipeline für die Analyse von Pulsaren“ (ML-PPA). Das Software-Framework stellt sich der Herausforderung, Pulsare aus großvolumigen Datenströmen zu identifizieren. Es wurde vom PUNCH4NFDI-Konsortium entwickelt, um sich der Herausforderung der Identifizierung von Pulsaren während der Datenerfassungsphase zu stellen. ML-PPA bietet unter anderem die Möglichkeit, synthetisierte Pulsardaten zu erstellen. Abbildung 3.1 zeigt den Vergleich zwischen realen Pulsardaten und synthetisierten Pulsardaten des Frameworks. Das Framework ist in der Lage, realistische Daten präzise zu rekonstruieren.

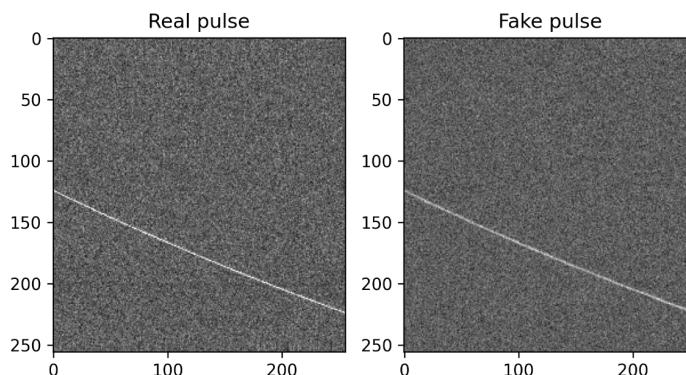


Abbildung 3.1: Vergleich eines echten Pulses vom Crab Pulsar und einem synthetisierten Pulse des ML-PPA Frameworks [3]. Die y-Achse entspricht der Frequenz und die x-Achse der Zeit in ms.

Zunächst wird ein Signal gemäß Gleichung 2.1 erstellt. Die Parameterwahl für DM, f_{LO} und f_{HI} sind hierbei entscheidend. Für die Erstellung der Datensätze wurden die DM-Werte aus dem ATNF Katalog entnommen, der Einträge zu über 3700 Pulsaren enthält [16]. Der Frequenzbereich wurde auf 1,21GHz bis 1,53GHz festgelegt, um eine umfassende Abdeckung zu gewährleisten. Anschließend wird dem erzeugten Signalbild ein Hintergrundrauschen hinzugefügt, das Gaußsches Rauschen enthält. Dieses Rauschen simuliert die zufälligen

Störungen, die in realen Messungen auftreten können, und ermöglicht eine realistischere Nachbildung der Bedingungen, unter denen die Signale erkannt werden müssen. Abbildung 3.2 veranschaulicht, wie das synthetisierte Signal vor und nach der Hinzufügung des Rauschens aussieht.

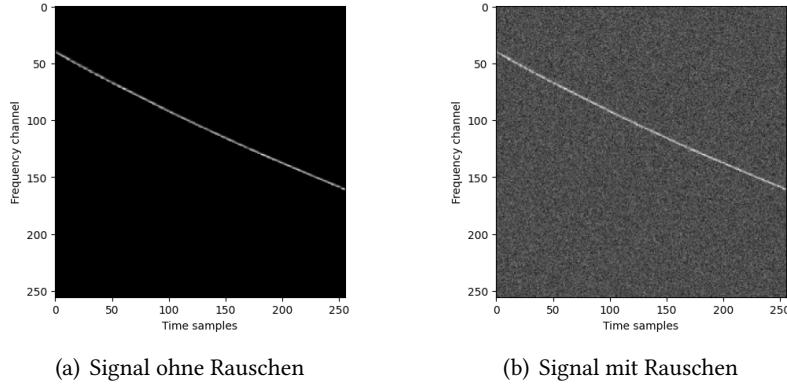


Abbildung 3.2: Darstellung des Prozesses zur Erzeugung synthetischer Pulsarsignale. a) zeigt das synthetisierte Signal ohne Hintergrundrauschen, b) illustriert das gleiche Signal mit Zugabe von Gaußschem Rauschen.

Das Framework bietet neben der Erstellung synthetizierter Pulsardaten ebenfalls die Erstellung von Breitband- (englisch: „Broadband Radio Interferences“, BBRFI) und Schmalband-Radiointerferenzen (englisch: „Narrowband Radio Interferences“, NBRFI). Diese Interferenzen stellen Störungen dar, die in Radioteleskopmessungen auftreten können. BBRFI sind Störungen, die in kurzer Zeit über den gesamten Frequenzbereich auftreten. NBRFI hingegen treten zu bestimmten Frequenzen oder engen Frequenzbereichen auf und bleiben über einen längeren Zeitraum bestehen. Durch die Integration dieser Signale lässt sich ein vielfältiger Datensatz erstellen.

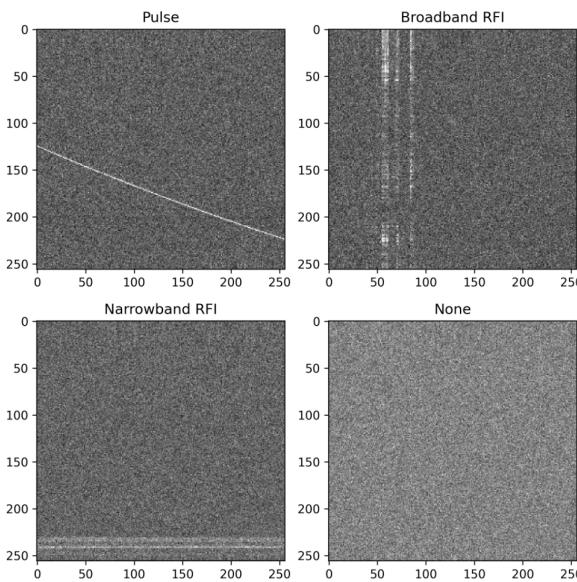


Abbildung 3.3: Beispiel von realen Beobachtungen des Crab Pulsars mit dem Radioteleskop Effelsberg [3].

Beispiele der verschiedenen Signalklassen, einschließlich der Klasse ohne Signal sind in Abbildung 3.3 zu sehen. Die Klasse „None“ stellt hierbei nicht das völlige Fehlen eines Signals dar, sondern dass kein Signal eindeutig identifiziert wurde. Um unterschiedliche Anwendungen abzudecken, wurden die Bildgrößen auf 32x32 und 256x256 Pixel festgelegt.

3.2. Angewendete Filter

Dieser Abschnitt stellt die verwendeten Filter vor und erläutert, wie sie die erste Ableitung eines Bildes approximieren. Ziel ist es, die charakteristische Signatur eines Pulsars, die als helle Kurve im Bild erscheint, durch die hier vorgestellten Kantenerkennungsfilter deutlicher hervorzuheben. Auf diese Weise sollen die relevanten Merkmale des Pulsars besser erkennbar gemacht und identifiziert werden.

3.2.1. Standard-Filter

Die Initialisierung der Gewichte ist ein entscheidender Schritt beim Aufbau neuronaler Netzwerke, da sie einen direkten Einfluss auf die Konvergenzgeschwindigkeit und die Lernfähigkeit des Netzwerks hat. Eine effektive Gewichtsinitialisierung trägt zu einem optimierten Lernprozess und zur Vermeidung von Problemen, wie dem Verschwinden oder Explodieren von Gradienten, bei.

Die He-Initialisierung wurde speziell entwickelt, um die Probleme der Gewichtsinitialisierung in tiefen neuronalen Netzwerken zu adressieren, insbesondere bei der Verwendung der ReLU-Aktivierungsfunktion. Diese Methode bietet eine verbesserte Anfangsverteilung der Gewichtswerte für jede Schicht des Netzwerks. Sie können entweder von einer Normalverteilung \mathcal{N} oder einer Gleichverteilung \mathcal{U} hervorgehen:

$$W \sim \mathcal{N}(0, \sqrt{\frac{2}{n_{in}}}) \quad W \sim \mathcal{U}(-\sqrt{\frac{6}{n_{in}}}, \sqrt{\frac{6}{n_{in}}}) \quad (3.1)$$

Hierbei steht n_{in} für die Anzahl der eingehenden Verbindungen zur jeweiligen Schicht. Durch diese gezielte Skalierung der Gewichtswerte wird sichergestellt, dass die Varianz der Aktivierungen stabil bleibt und das Netzwerk effektiver lernen kann.

Diese Methode der Gewichtsinitialisierung hat sich in vielen Netzwerkanwendungen bewährt, vor allem, da ReLU eine der am häufigsten genutzte Aktivierungsfunktion ist. Daher hat sich die He-Initialisierung zu einer Art „Standard“ etabliert, die in vielen Machine-Learning Frameworks und Bibliotheken als Voreinstellung oder empfohlene Methode zur Gewichtsinitialisierung verwendet wird.

Da es sich allerdings nach wie vor um eine zufällige Gewichtsinitialisierung handelt, benötigt das Netzwerks Zeit, um aus den zufällig Anfangswerten geeignete Werte zu entwickeln.

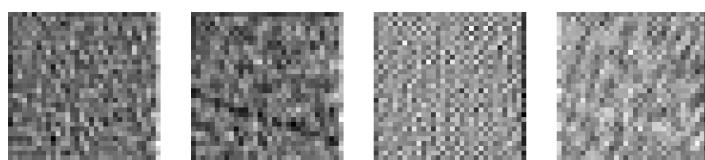


Abbildung 3.4: Beispiel für die Anwendung der Faltung mit vier zufällig initialisierten Filter auf ein 32x32 Bild eines Pulsars.

Abbildung 3.4 zeigt das Ergebnis der Faltung mit vier zufällig initialisierten Filtern anhand der He-Initialisierung auf ein Pulsarbild der Größe 32x32. Die anfänglich zufälligen Filter können bereits erste Merkmale erkennen, jedoch ist es auch möglich, dass sie keine relevanten Merkmale identifizieren oder die erkannten Merkmale nicht optimal zurückgeben. Diese Filter müssen durch das Trainieren des Netzwerks zunächst noch an die gegebenen Daten angepasst werden. Daher wird in dieser Arbeit untersucht, ob die Leistung eines CNNs verbessert werden kann, wenn bereits zu Beginn die relevanten Merkmale besser erkannt werden.

3.2.2. Prewitt-Filter

Der Prewitt-Filter ist ein einfacher Kantenerkennungsfilter, der in der Bildbearbeitung zur Kantenerkennung verwendet wird. Er gehört zur Klasse der Gradientenfilter und basiert auf der Berechnung der Ableitung der Pixelintensität in horizontaler und vertikaler Richtung [17]:

$$\begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}$$

Prewitt X

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix}$$

Prewitt Y

Die Koeffizienten der Filter summieren sich zu null, wodurch in Bereichen mit konstanter Intensität das Ergebnis ebenfalls null ist. Dabei handelt es sich um eine charakteristische Eigenschaft für einen Ableitungsoperator [18].

Da es sich bei der Pixelintensität um diskrete Werte handelt, ist es nicht möglich, die erste Ableitung zu berechnen. Bei Anwendung der Prewitt-Filter mit der Faltungsoperation aus Gleichung 2.5 lässt sich die erste Ableitung in sowohl horizontaler als auch vertikaler Richtung approximieren. Die partielle Ableitung einer diskreten Funktion f kann durch die Zentraldifferenz approximiert werden [18]:

$$\frac{\partial f(x, y)}{\partial x} \approx f(x + 1, y) - f(x - 1, y) \quad \frac{\partial f(x, y)}{\partial y} \approx f(x, y + 1) - f(x, y - 1) \quad (3.2)$$

Die Faltung für ein Bild I mit dem Prewitt-X-Filter P_X an einem Pixel (i, j) im Ergebnisbild S gemäß Gleichung 2.5 ergibt:

$$\begin{aligned} S_x(i, j) = P_X * I &= (-1) \cdot I(i - 1, j - 1) + 0 \cdot I(i - 1, j) + 1 \cdot I(i - 1, j + 1) + \\ &\quad (-1) \cdot I(i, j - 1) + 0 \cdot I(i, j) + 1 \cdot I(i, j + 1) + \\ &\quad (-1) \cdot I(i + 1, j - 1) + 0 \cdot I(i + 1, j) + 1 \cdot I(i + 1, j + 1) \\ &= I(i - 1, j + 1) - I(i - 1, j - 1) + \\ &\quad I(i, j + 1) - I(i, j - 1) + \\ &\quad I(i + 1, j + 1) - I(i + 1, j - 1) \end{aligned} \quad (3.3)$$

Für ein Pixel (i, j) im Bild S steht i für die Zeilen, und somit für die vertikale Richtung, und j für die Spalten, und somit für die horizontale Richtung. Um das mit der Zentraldifferenz vereinen zu können, müssen daher die Variablen in Gleichung 3.2 vertauscht werden:

$$\frac{\partial f(y, x)}{\partial x} \approx f(y, x + 1) - f(y, x - 1) \quad \frac{\partial f(y, x)}{\partial y} \approx f(y + 1, x) - f(y - 1, x)$$

Somit entspricht, aufgrund der Gewichte des Prewitt-X-Filters, die Operation in Gleichung 3.3 der partiellen Ableitung des Bildes I in horizontaler Richtung. Da der Prewitt-X-Filter die Intensitätsänderungen entlang der horizontalen Achse erfasst, reagiert er besonders empfindlich auf vertikale Kanten im Bild.

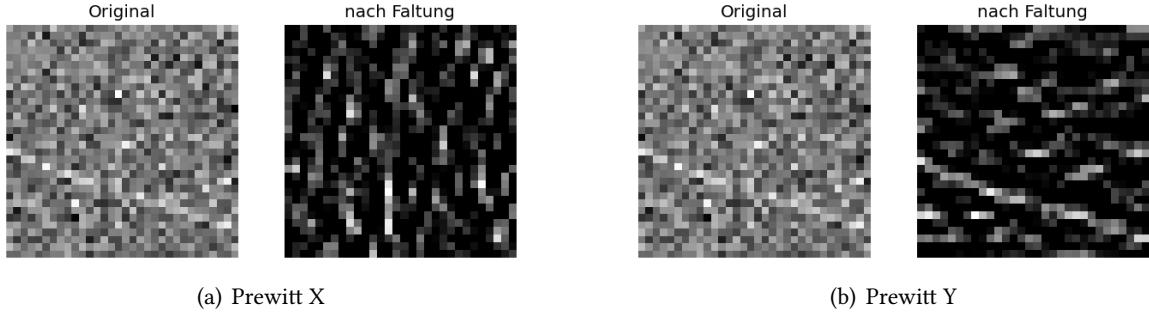


Abbildung 3.5: Beispiel für die Anwendung der Faltung mit den Prewitt-Filtern. Der Prewitt-Y-Filter erkennt aufgrund der Ableitung in vertikaler Richtung horizontale Kanten, wodurch der Puls hervorgehoben wird.

Abbildung 3.5 zeigt das Ergebnis der Faltung eines Pulsar-Bildes mit beiden Filtern. Dabei ist zu erkennen, dass der Prewitt-Y-Filter in der Lage ist, die Merkmale des Pulsars zu erfassen.

3.2.3. Sobel-Filter

Der Sobel-Filter ist ein weiterer Kantenfilter, der eine Erweiterung des Prewitt-Filters darstellt [17]. Dieser hat zusätzliche Gewichtungen bei den mittleren Koeffizienten, um die Kantenempfindlichkeit zu erhöhen und Rauschen zu reduzieren. Die Filter sind folgendermaßen definiert:

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

Sobel X

$$\begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

Sobel Y

Die Faltung eines Bildes I und dem Sobel-X-Filter ergibt:

$$\begin{aligned} S_x(i, j) = & I(i-1, j+1) - I(i-1, j-1) + \\ & 2I(i, j+1) - 2I(i, j-1) + \\ & I(i+1, j+1) - I(i+1, j-1) \end{aligned} \quad (3.4)$$

Da der Sobel-Filter die zentralen Pixel stärker berücksichtigt, haben die mittleren Pixelwerte im Bildausschnitt mehr Einfluss auf das Ergebnis, wodurch zufällige Schwankungen weniger stark ins Gewicht fallen und Kanten deutlicher hervorgehoben werden. Zusätzlich trägt die Gewichtung dazu bei, Rauschen im Bild zu reduzieren. In Bildern zeigt sich Rauschen als kleine, zufällige Intensitätsschwankungen, die lokal stark variieren können.

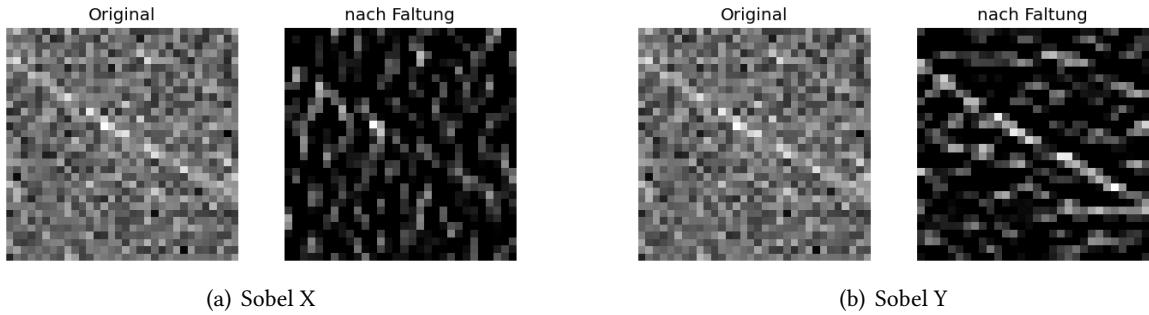


Abbildung 3.6: Beispiel für die Anwendung der Faltung mit den Sobel Filter. Der Sobel-Y-Filter erkennt aufgrund der Ableitung in vertikaler Richtung horizontale Kanten, wodurch der Puls hervorgehoben wird.

Abbildung 3.6 zeigt das Ergebnis der Faltung eines Bildes mit beiden Filtern. Dabei ist zu erkennen, dass der Sobel-Y-Filter in der Lage ist, die Merkmale des Pulsars zu erfassen.

3.2.4. Kirsch Kompass Maske

Die Kirsch Kompass Maske besteht aus acht Filtern und ermöglicht die Erkennung von Kanten in verschiedenen Richtungen. Die Sobel- und Prewitt-Filter wirken am stärksten bei horizontalen und vertikalen Kanten. Bei diesen Filtern wird eine Nord- oder Südkante als vertikale Kante angesehen. Die Kirsch-Filter hingegen unterscheiden zwischen den beiden Kanten anhand der Richtung der Intensitätsübergänge [18].

$$\max \left[1, \max_{i=0}^7 |5(a_i + a_{i+1} + a_{i+2}) - 3(a_{i+3} + a_{i+4} + a_{i+5} + a_{i+6} + a_{i+7})| \right] \quad (3.5)$$

Für einen beliebigen Punkt p werden die Helligkeitsunterschiede in der Nachbarschaft mithilfe der so genannten „Kontrastfunktion“ (siehe Gleichung 3.5) berechnet [19]. Sie dient dazu, den Kontrast in verschiedenen Richtungen zu ermitteln, um Kanten im Bild zu erkennen. Der Kontrast beschreibt den Intensitätsunterschied benachbarter Pixeln, daher treten Kanten an den Stellen auf, an denen dieser Unterschied am ausgeprägtesten ist. Durch die Berechnung des Maximums der Funktion wird die Richtung mit der stärksten Helligkeitsänderung identifiziert, was der Richtung der Kante entspricht. Dabei werden die Indizes Modulo 8 ausgewertet, um die umliegenden Punkte korrekt zu berücksichtigen. Die Nachbarschaft für p sei hierbei wie folgt nummeriert:

$$\begin{pmatrix} a_0 & a_1 & a_2 \\ a_7 & p & a_3 \\ a_6 & a_5 & a_4 \end{pmatrix}$$

Kirsch's Ansatz besteht darin, die Intensitätsänderung durch Faltung mit allen acht Filtern zu berechnen. Der höchste Wert, der dabei an einem bestimmten Punkt entsteht, gibt die Stärke der Kante an diesem Punkt an. Die Richtung der Kante wird dann durch den Filter mit dem höchsten Faltungswert festgelegt. Aus Gleichung 3.5 resultieren die folgenden Filter:

$\begin{pmatrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{pmatrix}$ <p>Kirsch N</p>	$\begin{pmatrix} -3 & -3 & -3 \\ -3 & 0 & 5 \\ -3 & 5 & 5 \end{pmatrix}$ <p>Kirsch NE</p>	$\begin{pmatrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{pmatrix}$ <p>Kirsch E</p>	$\begin{pmatrix} -3 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & 5 & -3 \end{pmatrix}$ <p>Kirsch SE</p>
$\begin{pmatrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{pmatrix}$ <p>Kirsch S</p>	$\begin{pmatrix} 5 & 5 & -3 \\ 5 & 0 & -3 \\ -3 & -3 & -3 \end{pmatrix}$ <p>Kirsch SW</p>	$\begin{pmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{pmatrix}$ <p>Kirsch W</p>	$\begin{pmatrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{pmatrix}$ <p>Kirsch NW</p>

Abbildung 3.8 zeigt die Ergebnisse der Faltung mit den verschiedenen Kirsch-Filters. Die Filter, die horizontale sowie diagonal von oben links nach unten rechts verlaufende Kanten erkennen, erfassen die Merkmale des Pulsars besonders gut.

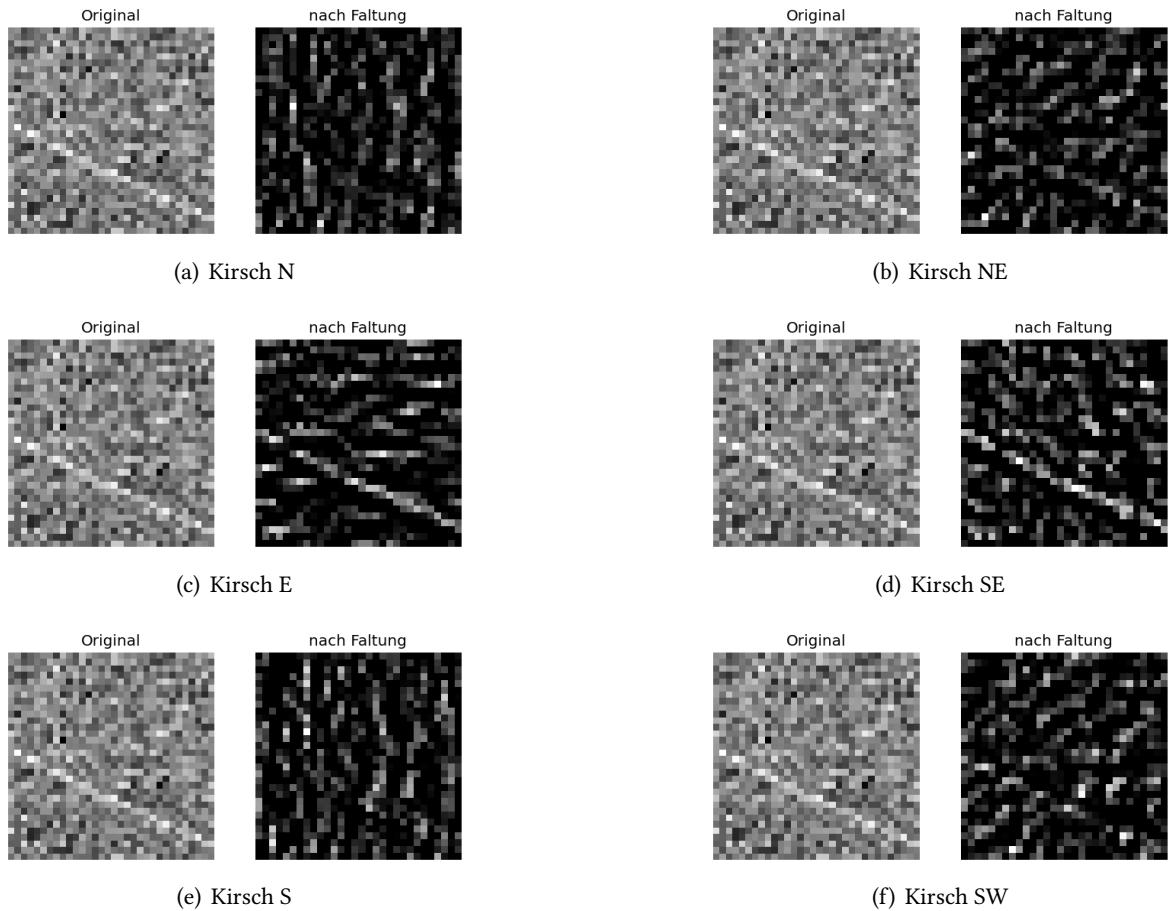


Abbildung 3.7: Beispiel für die Anwendung der Kirsch-Filter. Teil 1.

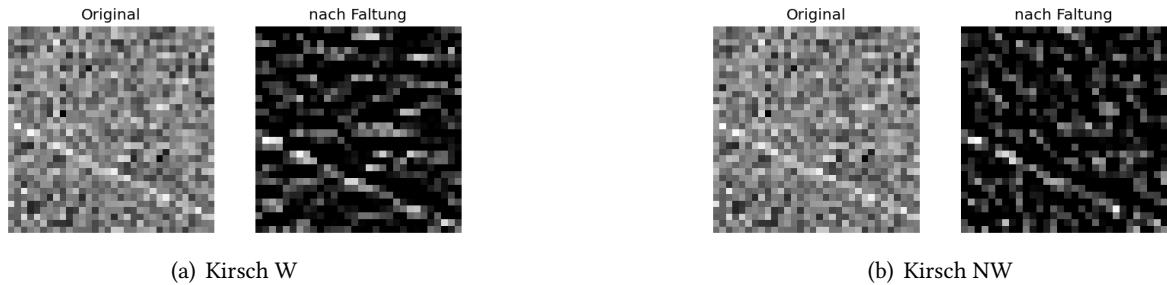


Abbildung 3.8: Beispiel für die Anwendung der Kirsch-Filter. Teil 2.

3.3. Netzwerk Architektur und Test Aufbau

Für die Implementierung des CNNs wurde das Deep-Learning-Framework PyTorch verwendet, da es durch seine benutzerfreundliche API sowie GPU-Unterstützung erlaubt, komplexe neuronale Netzwerke effizient zu entwickeln, zu trainieren und zu optimieren [20].

Um eine geeignete Architektur für den CNN zu finden, wurde ein Grid-Search-Hyperparameter-Tuning Programm geschrieben, das systematisch alle Permutationen verschiedener Werte der Hyperparameter testet. Dabei handelt es sich um die Parameter des neuronalen Netzes, die vor dem Trainingsprozess festgelegt werden müssen. Zu den getesteten Hyperparametern gehören die Anzahl der Schichten, die Anzahl der Filter in den Faltungsschichten sowie der Neuronen der vollständig verbundenen Schichten, die Filtergröße, die Lernrate des Optimierungsalgoritmus und die Batch-Größe.

Hyperparameter	Wert
Anzahl Faltungsschichten	2
Anzahl Filter (1. Schicht)	16
Anzahl Filter (2. Schicht)	64
Filtergröße	3x3
Anzahl vollständig verbundener Schichten (vor der Ausgabeschicht)	1
Anzahl Knoten	128
Lernrate	0,001
Batch-Größe	16

Tabelle 3.1: Ergebnis des Hyperparameter-Tuning Programms. Das Ergebnis beschreibt eine geeignete CNN Architektur für die hier verwendeten Datensätze.

Tabelle 3.1 zeigt die Ergebnisse des Programms. Es ist wichtig zu beachten, dass aufgrund der Vielzahl möglicher Hyperparameter-Kombinationen nur eine begrenzte Anzahl von Konfigurationen getestet werden konnte. Mit den gewählten Werten konnte dennoch eine effektive Netzwerkarchitektur gefunden werden, die in den durchgeföhrten Tests gute Ergebnisse erzielte.

Als Optimierungsalgoritmus wurde der Adam-Optimizer verwendet, da er durch seiner Verwendung adaptiver Lernraten und die Berücksichtigung des Mittelwerts und der Varianz der Gradienten zu einer

effizienteren und stabileren Konvergenz führt.

Für die Fehlerberechnung wurde die Kreuzentropie-Verlustfunktion verwendet, die sich besonders gut für Klassifikationsprobleme eignet. Sie quantifiziert die Abweichung zwischen den vom Netzwerk vorhergesagten Wahrscheinlichkeiten und der tatsächlichen Klasse eines Datenpunkts. Die Softmax-Funktion transformiert die Ausgaben des Netzwerks in Wahrscheinlichkeiten, sodass jede Klasse eine Wahrscheinlichkeit zugewiesen bekommt. In PyTorch ist die Softmax-Aktivierungsfunktion in die Kreuzentropie-Verlustfunktion integriert. Dies erleichtert die Netzwerkimplementierung, da die Kreuzentropie-Verlustfunktion in PyTorch sowohl die Berechnung der Wahrscheinlichkeiten als auch des Fehlers in einem Schritt übernimmt.

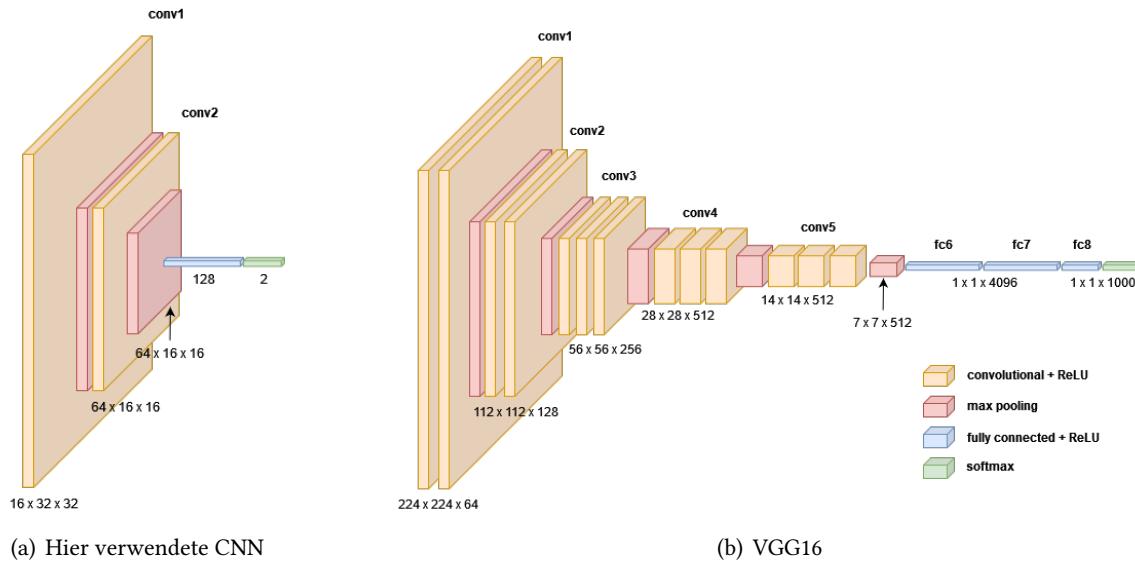


Abbildung 3.9: (a) stellt die Architektur des genutzten CNNs dar. Durch Padding bleibt die Größe des Bildes nach der Faltung unverändert. Aufgrund des 2x2 Max-Pooling halbiert sich die Größe der Feature Map. (b) stellt die Architektur des VGG16-Modells dar, das aufgrund seiner Tiefe für das hier verwendete Problem allerdings ungeeignet ist.

Die Architektur des CNNs ist in Abbildung 3.9 (a) dargestellt. Sie besteht aus zwei Faltungsschichten zur Merkmalsextraktion, die jeweils direkt von der ReLU-Aktivierungsfunktion gefolgt werden, mit dem das Netzwerk komplexe Muster erlernen kann. Anschließend folgen Max-Pooling-Schichten, die die Dimensionen der Feature-Maps reduzieren und die wichtigsten Merkmale extrahieren. Am Ende befinden sich zwei vollständig verbundene Schichten zur Klassifizierung der Daten aus den extrahierten Merkmalen.

Ursprünglich wurde für das ML-PPA-Frameworks das VGG16-Modell eingesetzt (siehe Abbildung 3.9 (b)) [21]. Die Architektur dieses Netzwerks wurde speziell für die Bildklassifizierung auf dem ImageNet-Datensatz entwickelt, der über 14 Millionen Bilder umfasst, die in mehr als 20.000 Kategorien unterteilt sind. Dieser dient als Standard-Benchmark für die jährliche „ImageNet Large Scale Visual Recognition Challenge“ (ILSVRC), bei der die Leistung von Bildklassifikations- und Objekterkennungsalgorithmen auf einer Teilmenge des Datensatzes mit etwa 1.000 Kategorien bewertet werden. Das Netzwerk ist aufgrund seiner tiefen Architektur in der Lage, besonders detaillierte und komplexe Merkmale aus Bildern zu extrahieren.

Für die vom ML-PPA erstellten und den in dieser Arbeit verwendeten Datensätze erwies sich VGG16 allerdings als redundant, da die verwendeten Daten deutlich weniger komplex und nur in 2 bzw. 4 Kategorien unterteilt sind. Die Anwendung des VGG16-Modells führte zu einem übermäßig hohen Rechenaufwand

und langen Trainingszeiten. Das Training des Netzwerks auf den ersten Datensatz mit einer Bildgröße von 256x256 benötigte etwa 20 Sekunden pro Epoche, wobei in einem vollständigen Testlauf zwei Modelle über 50 Epochen trainiert wurden. Das hätte in etwa 33 Minuten in Anspruch genommen. Da die Tests jeweils 10 Mal wiederholt werden, hätte dies zu einer Gesamtzeit von etwa $5\frac{1}{2}$ Stunden pro Test geführt. Diese langwierigen Testzeiten sind angesichts der relativ einfachen Struktur der verwendeten Datensätze nicht gerechtfertigt und verdeutlichen, dass das VGG16-Modell für diese spezifische Aufgabe ungeeignet ist. Daher bieten sich weniger komplexe Netzwerke deutlich mehr an.

Für die Tests wurden zwei Datensätze erstellt: Der erste enthält ausschließlich Pulsardaten und Daten ohne Signal, während der zweite zusätzlich Radiointerferenzsignale umfasst. Um unterschiedliche Anwendungen abzudecken, wurden die Bildgrößen auf 32x32 und 256x256 Pixel festgelegt. Kleinere Bilder reduzieren den Rechenaufwand und den Speicherbedarf, wodurch die verfügbaren Ressourcen effizienter genutzt werden und die Modellleistung potenziell verbessert werden kann. Allerdings führt dies zu einem Verlust an Detailgenauigkeit und Informationsgehalt der Bilder. Größere Bilder bieten mehr Details, erfordern jedoch mehr Rechenleistung und Speicher und können dementsprechend zu längeren Trainingszeiten führen. Die Evaluierung beider Bildgrößen ermöglicht es, fundierte Aussagen darüber zu treffen, ob und unter welchen Umständen es sinnvoll ist, die hier vorgestellten Methoden anzuwenden. Es wird zudem ein Rauschwert von 40% verwendet, um realen Bedingungen nahe zu sein.

Die Daten wurden in 80% Trainings- und 20% Testdaten aufgeteilt, um das Netzwerk auf einem Teil der Daten zu trainieren und anschließend seine Generalisierungsfähigkeit auf einem separaten, nicht zum Training verwendeten Datensatz zu testen. Die Leistung des CNNs wurde anhand der Genauigkeit (siehe Gleichung 3.6) bewertet, die angibt, wie viele der Vorhersagen des Netzwerks von den Testdaten korrekt sind.

Die Filter wurden in einer separaten Schicht am Anfang des CNNs implementiert, um deren Einfluss auf die Erkennungsgenauigkeit zu untersuchen. Das Training umfasste 50 Epochen wobei jeder Test 10 Mal wiederholt wurde, um anschließend den Mittelwert der Genauigkeit zu bestimmen.

$$\text{Genauigkeit} = \frac{\text{Anzahl der korrekt klassifizierten Beispiele}}{\text{Gesamtanzahl der Beispiele}} \quad (3.6)$$

Im Rahmen der Tests wurde die Leistung des CNNs untersucht, indem vordefinierte Filter verwendet wurden, die im Trainingsverlauf unverändert blieben, und gemeinsam mit dem Netzwerk optimiert wurden. Dadurch wird eine vergleichbare Analyse ermöglicht, um zu beurteilen, ob die Verwendung vordefinierter Filter ausreicht oder ob die Optimierung dieser Filter im Training zu besseren Ergebnissen führen könnte.

4 Ergebnis

In diesem Kapitel werden die Testergebnisse präsentiert. Zunächst werden die Resultate der Filteranwendung auf den ersten Datensatz vorgestellt, der ausschließlich Daten zu Pulsaren und Nicht-Signalen enthält. Anschließend werden die Ergebnisse für den erweiterten Datensatz präsentiert, der zusätzlich Radiointerferenzen umfasst. Zum Abschluss erfolgt eine Prüfung des Canny-Algorithmus auf einem angepassten Datensatz.

4.1. Erkennung von Pulsaren

Da die Signale eines Pulsars in einer Frequenz-Zeit-Messung eine charakteristische helle Kurve darstellen, bieten sich Kantenerkennungsfilter (siehe Abschnitt 3.2) zur verbesserten Identifizierung von Pulsaren an. Daher werden im folgenden Abschnitt die vorgestellten Filter in dem CNN angewendet, um die Genauigkeit (siehe 3.6) auf die Testdaten des Datensatzes mit ausschließlich der Klassen Pulsar und Nicht Signal zu analysieren.

Bildgröße	32x32	
Filter	Prewitt X	Prewitt Y
Genauigkeit	91,3%	97,25%

Tabelle 4.1: Ergebnisse der Prewitt-Filter bei separater Anwendung, ohne Veränderung durch Training.

Bildgröße	32x32	
Filter	Sobel X	Sobel Y
Genauigkeit	91,7%	96,25%

Tabelle 4.2: Ergebnisse der Sobel-Filter bei separater Anwendung, ohne Veränderung durch Training.

Bildgröße	32x32			
Filter	Kirsch N	Kirsch E	Kirsch S	Kirsch W
Genauigkeit	89,37%	97,6%	89,7%	98%
Filter	Kirsch NE	Kirsch SE	Kirsch SW	Kirsch NW
Genauigkeit	92,1%	96,25%	92,25%	95,87%

Tabelle 4.3: Ergebnisse der Kirsch-Filter bei separater Anwendung, ohne Veränderung durch Training.

In den Tabellen 4.1, 4.2 und 4.3 sind die Ergebnisse bei Anwendung der einzelnen Prewitt-, Sobel- und Kirsch-Filter auf 32x32 Bilder ohne zusätzliche Modifikation durch das Training des Netzwerks zu sehen. Die Ergebnisse zeigen, dass insbesondere Filter, die horizontale und diagonale Kanten von oben links nach unten rechts erkennen, höhere Genauigkeitswerte erzielen, da sie die Struktur des Pulsarsignals besser extrahieren können als Filter, die andere Kantenrichtungen erkennen.

Bildgröße		32x32				256x256			
Genauigkeit	Filter	Prewitt	Sobel	Kirsch	DF	Prewitt	Sobel	Kirsch	DF
		97,67%	97%	98,97%	97,7%	89,1%	88,55%	77,9%	89,6%
ohne Training	mit Training	97,7%	97,4%	99,15%		88,15%	88,65%	80,7%	

Tabelle 4.4: Ergebnisse der Kantenerkennungsfilter bei jeweils kombinierter Anwendung sowie der Default-Filter (DF).

Tabelle 4.4 zeigt die Genauigkeit bei jeweils kombinierter Anwendung der Filter, sowohl unverändert als auch gemeinsam mit dem Netzwerk trainiert. Die Kirsch-Filter sind dabei für 32x32 Bilder am besten in der Lage, die Merkmale der Daten zu extrahieren. Für 256x256 Bilder hingegen ist die Genauigkeit bei Anwendung der Kirsch-Filter geringer als bei den anderen Filtern. Durch die höheren Gewichtswerte sind sie empfindlicher für scharfe Kanten, dafür aber auch anfälliger für Rauschen oder kleine Variationen in den Kanten. Die Sobel-Filter führen aufgrund der höheren Gewichtung der mittleren Reihen bzw. Spalten eine integrierte Glättung aus. Das macht sie robuster als die Kirsch-Filter in der Erkennung von Pulsaren. Sie und die Prewitt-Filter sind dennoch nicht in der Lage, die größeren Bilder besser zu verarbeiten als die Default-Filter. Das Verändern der vordefinierten Filter durch das Training zeigt zudem keinen signifikanten Einfluss auf die Genauigkeit des CNNs.

Die vordefinierten Filter wurden mit der standardmäßig zufälligen He-Initialisierung der Filtergewichte verglichen, die als „Default-Filter“ bezeichnet werden. Hierbei wurde der CNN ohne die Schicht mit den vordefinierten Filtern ebenfalls 10 mal trainiert, um den Mittelwert der Genauigkeit zu erhalten. Dabei schwankte die Genauigkeit des CNNs bei zufälliger Initialisierung, aufgrund der Variabilität der zufälligen Gewichte. Die Genauigkeit für 32x32 Bilder betrug in einigen Fällen 94% und erreichte in anderen bis zu 99%. Die vordefinierten Filter erzielten hingegen bei jedem Durchlauf stabile Ergebnisse, wobei die Kirsch-Filter konstant die höchsten Werte aufwiesen.

Für 256x256 Bilder zeigte das Netzwerk mit den Default Filtern gelegentlich Schwierigkeiten, eine Klasse korrekt zu erkennen, wodurch die Genauigkeit auf unter 50% sank. Das kann auf zwei potentielle Probleme zurückgeführt werden. Zum einen könnte es sich um Überanpassung (englisch: „Overfitting“) handeln, wobei das Netzwerk zu stark an die Trainingsdaten angepasst wurde und dadurch Schwierigkeiten hat, neue, ungesehene Daten korrekt zu klassifizieren. Zum anderen könnte das „Dying-ReLU“-Problem [11] eine Rolle gespielt haben, bei dem einige Neuronen dauerhaft 0 ausgeben und somit keine nützlichen Gradienten für das Training liefern. Beide Probleme könnten durch die zufällige Initialisierung der Filtergewichte verursacht worden sein, die dazu führte, dass wichtige Merkmale der Bilder zu Beginn nicht richtig erfasst oder sogar negiert wurden. Diese Einzelfälle wurden aus der Bewertung ausgeschlossen.

Die Ergebnisse zeigen, dass die vordefinierten Filter bei der Erkennung von Pulsaren mit einer Bildgröße von 32x32 zu einer stabileren und verbesserten Leistung des CNNs führen. Für 256x256 Bilder erreichen die Prewitt- und Sobel-Filter ebenfalls stabile Ergebnisse, konnten aber keine Verbesserungen aufweisen.

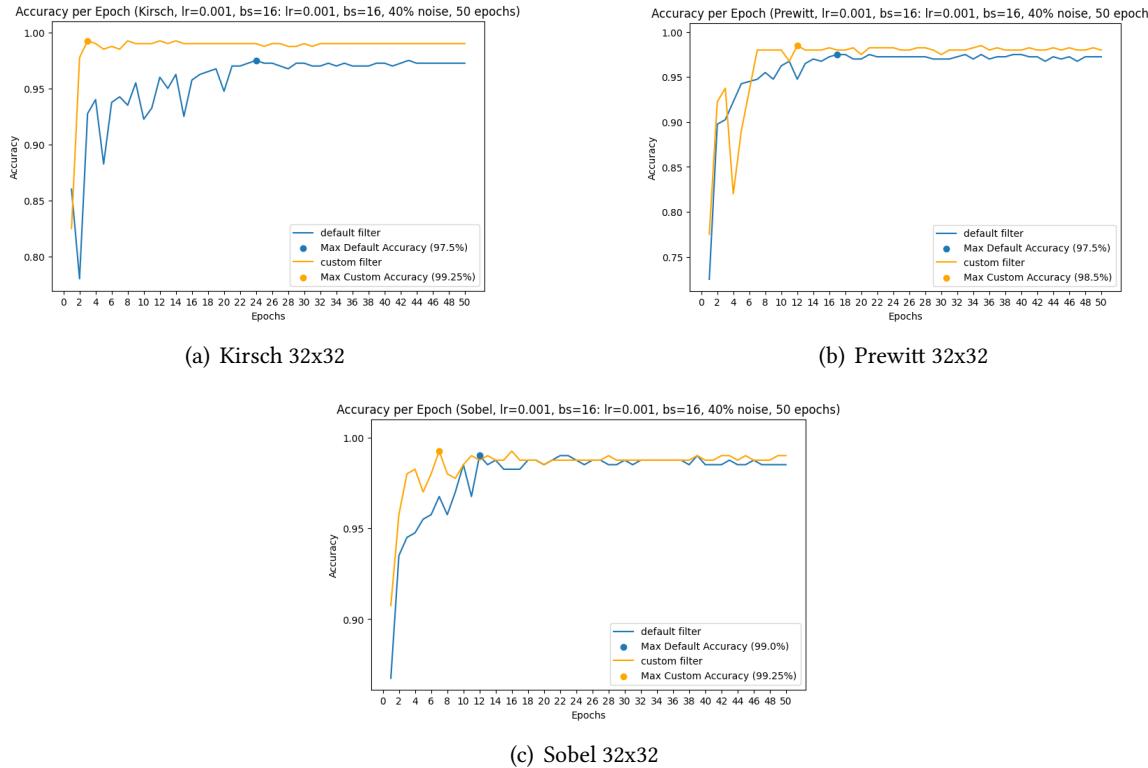


Abbildung 4.1: Genauigkeitsgraph auf den Testdaten in Abhängigkeit von der Anzahl der Epochen. In (a) wurden die Kirsch-Filter, in (b) die Prewitt-Filter und in (c) die Sobel-Filter auf 32x32 Bilder angewendet.

Abbildung 4.1 zeigt die Genauigkeit auf den Testdaten in Abhängigkeit von der Anzahl der Epochen bei Anwendung der Kirsch- (a), Prewitt-Filter (b) und Sobel-Filter (c) für 32x32 Bilder. Die blaue Kurve stellt die Genauigkeit pro Epoche bei Verwendung der Default Filter dar und die orangene Kurve zeigt die Genauigkeit pro Epoche bei Verwendung der vordefinierten Filter. Die höchste erreichte Genauigkeit beider Methoden ist jeweils durch eine Markierung hervorgehoben. Zu erkennen ist, dass bei Anwendung der vordefinierten Filter das Netzwerk schneller konvergiert. Der CNN ist bereits früh in der Lage, die relevanten Merkmale zu extrahieren. Im Gegensatz dazu weisen die Default-Filter Schwankung über die Epochen auf, wodurch sie erst spät konvergieren. Die Anwendung der Filter auf 256x256 Bilder resultierte in ähnlichen Verläufen, die allerdings eine geringere Genauigkeit erreichten und dementsprechend die Kurven nach unten verschoben sind.

Bildgröße	32x32	256x256
Filter Genauigkeit	Prewitt Y, Sobel Y, Kirsch E&SE&W&NW	Prewitt Y, Sobel Y, Kirsch E&SE&W&NW
ohne Training	99,15%	82,25%
mit Training	99%	81,45%

Tabelle 4.5: Ergebnisse der gemeinsamen Anwendung der Kantenerkennungsfilter Prewitt Y, Sobel Y, Kirsch E, Kirsch SE, Kirsch W und Kirsch NW.

Da die Filter, die horizontale sowie diagonale Kanten von oben links nach unten rechts erkennen, am effektivsten bei der Extraktion der relevanten Merkmale waren, wurden sie in Kombination angewendet. Tabelle 4.5 zeigt jedoch, dass die Kombination der Filter Prewitt Y, Sobel Y, Kirsch E, Kirsch SE, Kirsch W und Kirsch NW für 256x256 Bilder zu einer geringeren Genauigkeit führt. Das kann aufgrund der nachteiligen Eigenschaften der Kirsch Filter bezüglich der hier verwendeten Daten begründet werden, die den CNN negativ beeinflussen. Die Anpassung der Filter durch das Training wirkt sich dabei nachteilig auf die Genauigkeit des CNNs aus.

Die Ergebnisse dieses Abschnitts zeigen, dass die Verwendung vordefinierter Filter bei der Klassifikation von Daten mit Pulsaren und Nicht-Signalen sowohl für kleinere als auch größere Bilder eine schnellere Konvergenz des CNNs ermöglicht. Die Genauigkeit des CNNs blieb stabil und erreichte für kleinere Bilder höhere Werte, allerdings zeigten die Filter Schwierigkeiten bei der Verarbeitung größerer Bilder. Besonders die Kirsch-Filter schnitten aufgrund ihrer Anfälligkeit für Rauschen am schlechtesten ab.

4.2. Erweiterter Datensatz

In diesem Abschnitt werden die Ergebnisse für das Testen auf einen Datensatz mit zusätzlichen Daten für Breitband- und Schmalband-Radiointerferenzen vorgestellt.

Breitband-Radiointerferenzen treten als eine vertikale Struktur im Bild auf, die allerdings nicht als durchgehende Linie interpretiert werden kann. Stattdessen handelt es sich um eine Anordnung zahlreicher heller Pixel, die sich vertikal erstrecken und deutlich vom mit Rauschen bedecktem Hintergrund abheben, jedoch keine kontinuierliche Linie bilden. Abbildung 4.2 zeigt eine beispielhafte Darstellung des Signals mit einer Bildgröße von 32x32 (a) und 256x256 (b).

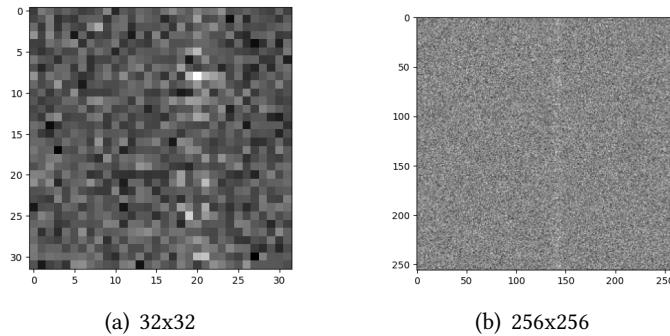


Abbildung 4.2: Beispiel eines BBRFI Signals mit einer Bildgröße von (a) 32x32 und (b) 256x256.

Schmalband-Radiointerferenzen hingegen kennzeichnen sich durch eine schmale horizontale Linie, die angebt, dass das Signal zu bestimmten Frequenzen oder schmalen Frequenzbereichen über längere Zeit erhalten bleiben [3]. Daher eignen sich für ihre Erkennung horizontale Kantenerkennungsfilter.

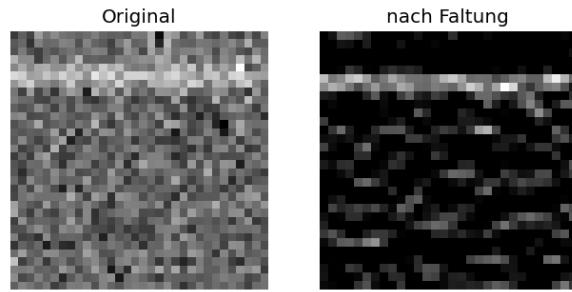


Abbildung 4.3: Ergebnisse der Faltung mit dem Sobel Y Filter auf ein Schmalband-Radiointerferenz Bild der Größe 32x32.

Abbildung 4.3 zeigt das Ergebnis der Faltung mit dem Sobel Y Filter auf ein beispielhaftes Bild der Größe 32x32 einer Schmalband-Radiointerferenz Messung. Dabei ist deutlich zu erkennen, dass der Filter in der Lage ist, das horizontale Signals klar hervorzuheben.

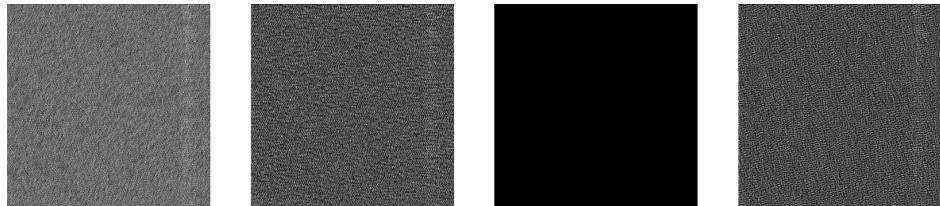


Abbildung 4.4: Ergebnisse der Faltung auf ein Breitband-Radiointerferenz Bild der Größe 256x256 mit vier Default Filtern.

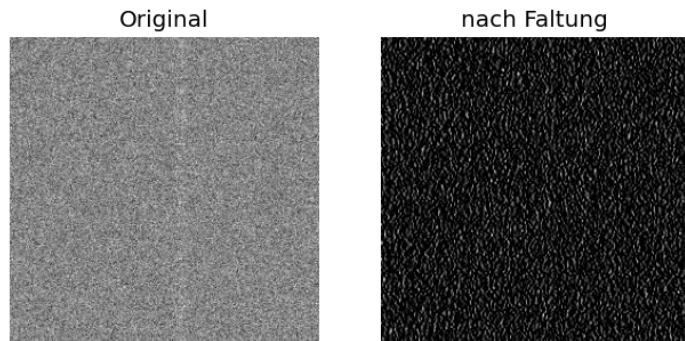


Abbildung 4.5: Ergebnisse der Faltung auf ein Breitband-Radiointerferenz Bild der Größe 256x256 mit dem Sobel X Filter.

In Abbildung 4.4 ist das Ergebnis der Faltung mit vier zufällig initialisierten Filtern zu sehen. Die Default Filter können, aufgrund ihrer zufälligen Gewichtsinitialisierung, eine Vielzahl von Merkmalen erfassen, die nicht auf eine bestimmte Richtung oder einen bestimmten Kantentyp spezialisiert sind. Dadurch sind einige in der Lage, die Merkmale des gegebenen Bildes zu erkennen. Die vollständig schwarze Abbildung ist das Resultat nach der ReLU-Aktivierungsfunktion aufgrund, durch die Faltung entstandener, negativer Werte.

Abbildung 4.5 veranschaulicht, dass vertikale Kantenerkennungsfilter nicht in der Lage sind, die Merkmale eines BBRFI-Bildes zu extrahieren. Die Filter haben Schwierigkeiten, diese Daten richtig zu klassifizieren,

da sie keine richtigen Kanten beinhalten. Die Folge dessen ist eine geringere Genauigkeit bezüglich dieser Signalklasse.

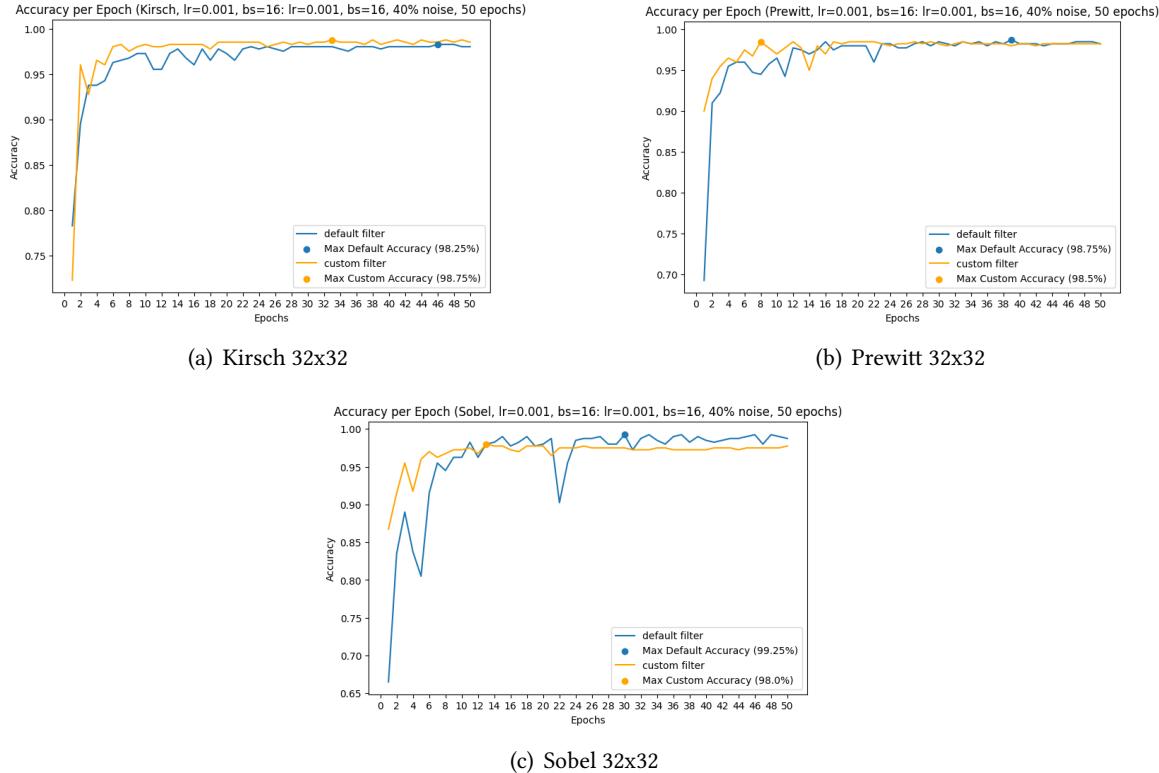


Abbildung 4.6: Genauigkeit auf den Testdaten in Abhängigkeit von der Anzahl der Epochen bei Anwendung der Kirsch-Filter (a) Prewitt-Filter (b) und Sobel-Filter (c) auf 32x32 Bilder.

Abbildung 4.6 zeigt, dass die Verwendung der vordefinierten Filter für 32x32 Bilder zu einer früheren Konvergenz führt, allerdings keine Verbesserungen aufweisen konnten. Für 256x256 Bilder waren die vordefinierten Filter nicht in der Lage, die Merkmale des Datensatzes ausreichend gut zu erkennen. Ihre Anwendung führte zu einer absteigenden Genauigkeit bei fortschreitender Epochenzahl.

Bildgröße		32x32				256x256			
Filter	Genauigkeit	Prewitt	Sobel	Kirsch	DF	Prewitt	Sobel	Kirsch	DF
ohne Training	97,9%	96,77%	98,97%	97,74%	97,74%	85,25%	82,8%	81,45%	90%
mit Training	97,4%	97,77%	98,87%			84,47%	84,52%	82,3%	

Tabelle 4.6: Ergebnisse der Kantenerkennungsfilter auf den erweiterten Datensatz bei jeweils kombinierter Anwendung sowie der Default-Filter (DF).

Tabelle 4.6 zeigt die Ergebnisse der gemeinsamen Anwendung der Filter sowohl unverändert als auch verändert. Für kleinere Bilder erweisen sich die vordefinierten Filter als geeignet bei der Merkmalsextraktion, wobei die Kirsch-Filter erneut die besten Ergebnisse liefern. Aufgrund des erhöhten Detail- und Informationsgehaltes sinkt die Genauigkeit bei größeren Bildern deutlich. Dadurch wirft unter anderem die

Verarbeitung der Breitband-Radiointerferenz-Daten Schwierigkeiten auf.

Die Ergebnisse dieses Abschnittes verdeutlichen, dass die vordefinierten Filter für kleinere Bilder eine frühere Konvergenz erreichten, jedoch keine wesentlichen Verbesserungen aufzeigen konnten. Für größere Bilder verschlechtert sich die Leistung des CNNs bei Anwendung der Filter, aufgrund ihrer Schwierigkeit die Radiointerferenz-Signale korrekt zu klassifizieren.

4.3. Anwendung des Canny-Algorithmus

Da die vordefinierten Filter bei größeren Bildern Schwierigkeiten hatten, die Daten richtig zu erkennen, wird der Canny-Algorithmus als Verbesserungsansatz angewendet. Dabei handelt es sich um ein leistungsfähiges Verfahren zur Kantendetektion in der Bildbearbeitung, das aus mehreren aufeinanderfolgenden Schritten besteht [22].

Zu Beginn wird das Bild geglättet, um Rauschen zu reduzieren und die Kantenerkennung zu verbessern. Dazu wird ein Gauß-Filter auf das Bild angewendet, der auf die Gaußsche Verteilungsfunktion basiert:

$$G(x, y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (4.1)$$

Hierbei sind x und y die Koordinaten relativ zur Mitte des Filters, und σ ist die Standardabweichung, die die Stärke der Glättung bestimmt. Durch die Faltung des Bildes mit dem Gauß-Filter wird Rauschen reduziert, wodurch eine stabilere und präzisere Kantenerkennung ermöglicht wird.

Nach der Glättung wird ein Gradientenfilter, wie beispielsweise der Sobel- oder Prewitt-Filter, angewendet, um die Kanten im Bild zu identifizieren.

Im nächsten Schritt erfolgt die Nicht-Maximum-Unterdrückung, um die Kanten im Bild dünn und präzise zu machen. Dabei werden alle Pixel, die keine lokalen Maxima entlang der Gradientenrichtung sind, unterdrückt. Die Gradientenrichtung gibt die Richtung der stärksten Änderung der Intensität an einem Punkt an. Dadurch werden nur die stärksten Kantenpunkte beibehalten, wodurch die Kantenbreite reduziert und die Kanten besser definiert werden.

Im letzten Schritt des Canny-Algorithmus wird eine Hysterese-Schwellenwertsetzung angewendet. Dafür werden zwei Schwellenwerte definiert, die die Kanten anhand ihrer Intensitätswerte bewerten und akzeptieren oder verwerfen.

In dieser Arbeit wird allerdings eine vereinfachte Variante des Canny-Algorithmus verwendet, der nur die ersten beiden Schritte beinhaltet. Durch die Glättung soll untersucht werden, ob die Kanten trotz des vorhandenen Rauschens im Bild besser erkannt werden können. Die Nicht-Maximum-Unterdrückung und die Hysterese-Schwellenwertsetzung werden bewusst weggelassen, da sie bei feinen Kanten und starkem Hintergrundrauschen negative Auswirkungen auf die Datenerkennung haben könnten, indem sie wichtige Kanten unterdrücken oder unterbrechen.

Zusätzlich wurde der erweiterte Datensatz angepasst, indem Breitband-Radiointerferenzen (BBRFI) nicht mehr in eine eigene Klasse eingeordnet, sondern zur Klasse der Nicht-Signale hinzugefügt wurden.

Diese Anpassung beruht auf der Beobachtung, dass BBRFI-Messungen nur schwach erkannt werden und daher die Unterscheidung in eine separate Klasse sie nicht ausreichend differenziert.

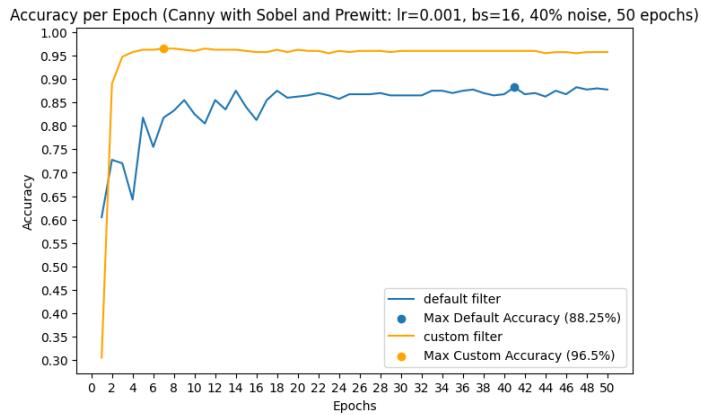


Abbildung 4.7: Genauigkeitsgraph bei Anwendung des Canny-Algorithmus mit dem horizontalen Prewitt- und Sobel-Filter auf den angepassten, erweiterten Datensatz

Abbildung 4.7 zeigt, dass die Verwendung des Canny-Algorithmus in Kombination mit den horizontalen Prewitt- und Sobel-Filtern zu einer deutlich schnelleren Konvergenz und besserem Ergebnis führt als der Default Filter.

Datensatz	Pulsar - Nicht Signal	Pulsar - Nicht Signal - BBRFI - NBRFI	Pulsar - Nicht Signal - NBRFI
Genauigkeit (Default Filter)	89,63	90%	90%
Genauigkeit (Canny-Algo)	84,95%	91,35%	94,92%

Tabelle 4.7: Ergebnisse der Kantenerkennungsfilter auf den angepassten, erweiterten Datensatz bei Anwendung des Canny-Algorithmus.

Die Ergebnisse in Tabelle 4.7 zeigen, dass besonders für den angepassten Datensatz die Anwendung des Canny-Algorithmus zu einer erhöhten Genauigkeit führt. Die zusätzliche Glättung des Bildes vor der Kantenerkennung zeigt positive Auswirkungen auf die Genauigkeit des CNNs.

5 Fazit

Diese Arbeit befasste sich mit der Optimierung der Erkennung von Pulsaren mithilfe von Convolutional Neural Networks (CNNs).

Das Ziel der Arbeit war es zu untersuchen, ob eine optimierte Filter-Initialisierung die Leistung von CNNs bei der Erkennung von Pulsaren verbessern kann. Dafür wurden drei spezielle vordefinierte Filter getestet: Prewitt, Sobel und Kirsch. Diese Filter sind auf die Detektion von Kanten ausgelegt, die als eine Änderung in der Intensität des Bildes sichtbar sind. Das erreichen sie durch die Approximation der ersten Ableitung der Pixelintensität des Bildes anhand der Faltungsoperation.

Die Filter wurden als zusätzliche Schicht zu Beginn des CNNs implementiert, um eine initiale Kantenerkennung durchzuführen, bevor das Netzwerk weiter trainiert wurde. Sie blieben entweder unverändert oder wurden während des Trainings optimiert, um ihre Auswirkungen auf die Genauigkeit des CNNs zu bewerten. Das Training wurde auf zwei verschiedenen Datensätzen durchgeführt. Der erste Datensatz enthielt ausschließlich Daten zu Pulsaren und zur Abwesenheit eines Signals, der zweite Datensatz umfasste zusätzlich auch Daten mit Schmalband- (NBRFI) und Breitband-Radiofrequenzstörungen (BBRFI). Dabei wurden separate Datensätze mit Bildern der Größe 32x32 als auch 256x256 Bilder verwendet, um unterschiedliche Anwendungen abzudecken.

Die Ergebnisse zeigten, dass die vordefinierten Filter auf dem ersten Datensatz mit 32x32 Bildern eine stabilere und konsistenter Leistung im Vergleich zur zufälligen Initialisierung der Filtergewichte aufweisen konnten, und ebenfalls schneller konvergiert sind. Während Netzwerke mit zufälliger Initialisierung bei jedem Durchlauf unterschiedliche Genauigkeitswerte zwischen 94% und 99% erzielten, lieferten die vordefinierten Filter gleichmäßig hohe Ergebnisse. Besonders die Kirsch-Filter erreichten konstant eine Genauigkeit von 99%. Für 256x256 Bilder zeigten die Prewitt- und Sobel-Filter ebenfalls eine schnellere Konvergenz. Mit einer Genauigkeit von 89,1% bzw. 88,55% war ihre Leistung ähnlich zu den Default Filtern, die durchschnittlich eine Genauigkeit von 89,63% erzielten. Die Kirsch-Filter erzielten aufgrund ihrer Anfälligkeit für Rauschen für größere Bilder keine optimalen Ergebnisse.

Bei der Anwendung der vordefinierten Filter auf den erweiterten Datensatz zeigten sich keine signifikanten Verbesserungen in der Erkennungsgenauigkeit. Für 32x32-Bilder konnten die Filter schneller konvergieren und hohe Werte erreichen, jedoch führten sie nicht zu wesentlichen Leistungssteigerungen. Bei 256x256-Bildern verschlechterte sich ihre Leistung, da sie die Merkmale der BBRFI-Daten nicht erfassen konnten. Dies verdeutlicht, dass die vordefinierten Filter zwar effektiv für die Pulsarerkennung sind, jedoch bei komplexeren Datensätzen mit zusätzlichen Störungen keine signifikanten Vorteile bieten.

Durch die Verwendung des Canny-Algorithmus und der Anpassung des erweiterten Datensatzes mit 256x256 Bildern, durch die Zusammenführung der Breitband-Radiointerferenzen zu den Nicht-Signalen,

erreichte der CNN eine verbesserte Genauigkeit. Die Glättung des Bildes zur Rauschunterdrückung zeigte positive Auswirkungen auf die Leistung des CNNs, da das Netzwerk deutlich schneller konvergierte. Das verdeutlicht, dass sich diese Methode zur Pulsarerkennung eignet.

Zusammenfassend zeigt sich, dass die Implementierung vordefinierter Kantenerkennungsfilter in Convolutional Neural Networks (CNNs) die Erkennung von Pulsaren bei kleineren Bildgrößen stabilisieren und verbessern kann. Die Filter trugen zu einer gleichmäßigeren Leistung über mehrere Trainingsläufe bei, und beschleunigten den Trainingsprozess, indem sie bereits frühzeitig die relevanten Merkmale der Pulsarsignale effektiv extrahieren konnten. Bei größeren Bildern waren die Filter ebenfalls in der Lage, eine frühere Konvergenz zu erreichen, zeigten jedoch keine signifikanten Verbesserungen in der Erkennungsgenauigkeit. Ihre Anwendung auf den erweiterten Datensatz führte ebenfalls zu keiner wesentlichen Steigerung der Genauigkeit, was auf die Begrenztheit dieser vordefinierten Filterstrukturen bei komplexeren Datensätzen hinweist. Durch die Anpassung des erweiterten Datensatzes und der Verwendung des Canny-Algorithmus, konnten die Filter eine Leistungssteigerung erzielen. Die vorherige Glättung des Bildes zeigte positive Auswirkungen und trug zur Verbesserung der Filterleistung bei.

Diese Arbeit verdeutlicht, dass vordefinierte Filter in CNNs eine stabile und effiziente Lösung zur verbesserten Erkennung von Pulsaren bieten können. Dadurch konnte das Ziel der Arbeit erreicht werden, die Erkennung von Pulsaren in verschiedenen Anwendungsfällen durch optimierte Filter-Initialisierung zu verbessern.

6 Literatur

- [1] Hendrik van Hees. “Grundlagen zum Pulsar-Timing”. 2017.
- [2] Andreas Müller. *10 Dinge, die Sie über Gravitationswellen wissen wollen*. S. 124. Springer Berlin, Heidelberg, 2017.
- [3] Andrei Kazantsev u. a. “ML-based Pipeline for Pulsar Analysis (ML-PPA)”. 2023.
- [4] *PUNCH4NFDI*. <https://www.punch4nfdi.de/>. zuletzt Abgerufen am 20. August 2024.
- [5] Michael A. Nielsen. *Neural Networks and Deep Learning*. <http://neuralnetworksanddeeplearning.com/index.html>. Determination Press, 2016.
- [6] M.V. Narkhede, P.P. Bartakke und M.S. Sutaone. “A review on weight initialization strategies for neural networks”. 2021.
- [7] Benjamin Hack. “Auswirkungen des Sonnenwinds auf Pulsar Timing Messungen”. 2021.
- [8] Y. Bengio, P. Simard und P. Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE Transactions on Neural Networks* 5.2 (1994), S. 157–166. doi: 10.1109/72.279181.
- [9] Diederik P. Kingma und Jimmy Lei Ba. “Adam: A Method for Stochastic Optimization”. 2017.
- [10] Yann Lecun, Patrick Haffner und Y. Bengio. “Gradient-Based Learning Applied to Document Recognition”. 1998.
- [11] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. <http://www.deeplearningbook.org>, S. 326-366. MIT Press, 2016.
- [12] *MaxpoolSample2*. <https://computersciencewiki.org/index.php/File:MaxpoolSample2.png>. zuletzt Abgerufen am 17. August 2024. 2018.
- [13] Saad Albawi, Tareq Abed Mohammed und Saad Alzawi. “Understanding of a Convolutional Neural Network”. In: Aug. 2017. doi: 10.1109/ICEngTechnol.2017.8308186.
- [14] Bernd Jähne. *Digitale Bildverarbeitung und Bildgewinnung*. Springer Berlin, Heidelberg, 2012.
- [15] Addison Sears-Collins. *How the Sobel Operator Works*. <https://automaticaddison.com/how-the-sobel-operator-works/>. zuletzt Abgerufen am 10. August 2024. 2019.

- [16] L. Toomey G. Hobbs R. N. Manchester und A. Kapur. *ATNF Pulsar Catalogue*. <https://www.atnf.csiro.au/people/pulsar/psrcat/>. zuletzt Abgerufen am 10. August 2024.
- [17] Zsuzsanna Huber. "Versteckte Codierung von Bildinformationen". S. 20-22. Universität Passau, 2018.
- [18] Rafael C. Gonzalez und Richard E. Woods. *Digital Image Processing*. S. 184-187, 720, 724. Pearson, 2018.
- [19] Russel A. Kirsch. "Computer Determination of the Constituent Structure of Biological Images". In: (1971).
- [20] Facebook AI Research. *PyTorch*. <https://pytorch.org/>. zuletzt Abgerufen am 17. August 2024.
- [21] Karen Simonyan und Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV]. URL: <https://arxiv.org/abs/1409.1556>.
- [22] John Canny. "A Computational Approach To Edge Detection". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on PAMI-8* (Dez. 1986), S. 679–698. doi: 10.1109/TPAMI.1986.4767851.

Abbildungsverzeichnis

2.1	Beispiel für die Ausbreitungsverzögerung eines Pulses von J1800+0534	3
2.2	Beispiel Aktivierung eines Knotens	4
2.3	Beispiel für ein neuronales Netz mit 2 verborgenen Schichten.	5
2.4	Beispiel Faltung eines Filters	7
2.5	Beispiel für 2x2 Max-Pooling	8
2.6	Beispiel einer Kante im Bild	9
2.7	Pixelintensitätgraph mit Steigung	10
3.1	Vergleich eines echten Pulses mit einem Synthetisierten	11
3.2	Signal ohne und mit Rauschen	12
3.3	Beispiel von realen Beobachtungen	12
3.4	Beispiel für Faltung zufälliger Filter	13
3.5	Beispiel für die Anwendung der Prewitt-Filter	15
3.6	Beispiel für die Anwendung der Sobel-Filter	16
3.9	CNN Architekturen	19
4.1	Genauigkeitsgraph der Filter auf den ersten Datensatz mit 32x32 Bilder	23
4.2	Beispiel BBRFI	24
4.3	Beispiel Ergebnis Sobel Y auf NBRFI Daten	25
4.4	Ergebnis Faltung auf BBRFI Daten mit Default Filter	25
4.5	Ergebnis Faltung auf BBRFI Daten mit Sobel X	25
4.6	Genauigkeitsgraph bei Anwendung der Filter auf den erweiterten Datensatz mit 256x256 Bilder	26
4.7	Genauigkeitsgraph bei Anwendung des Canny-Algorithmus auf den angepassten, erweiterten Datensatz	28

Bitte dieses Formular zusammen mit der Abschlussarbeit abgeben!

Erklärung zur Abschlussarbeit

gemäß § 35, Abs. 16 der Ordnung für den Bachelorstudiengang Informatik vom 17. Juni 2019:

Hiermit erkläre ich

(Nachname, Vorname)

Die vorliegende Arbeit habe ich selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst.

Ich bestätige außerdem, dass die vorliegende Arbeit nicht, auch nicht auszugsweise, für eine andere Prüfung oder Studienleistung verwendet wurde.

Zudem versichere ich, dass alle eingereichten schriftlichen gebundenen Versionen meiner vorliegenden Bachelorarbeit mit der digital eingereichten elektronischen Version meiner Bachelorarbeit übereinstimmen.

Frankfurt am Main, den

Unterschrift der/des Studierenden